VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Zachycení typického vytížení nad SQL Serverem

Capture of a Typical SQL Server Database Workload

2017 Bc. Tomáš Bauer

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Tomáš Bauer**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Zachycení typického vytížení nad SQL Serverem
Capture of a Typical SQL Server Database Workload

Jazyk vypracování: čeština

Zásady pro vypracování:

V SQL Serveru existuje celá řada nástrojů, které umožňuji zaznamenávat příkazy nad databází. Problémem těchto nástrojů je, že objem takto zaznamenaných dat může být nemalý a data vyžadují další zpracování. Cílem této práce je vytvoření nástroje, který umožní zaznamenat typické příkazy nad SQL Server databází, který tyto problémy mít nebude. Hlavní vlastnosti/funkce tohoto nástroje budou:
1. Možnost spuštění nástroje jako procedury na SQL Serveru s minimem oprávnění.
2. Nástroj bude dle nastavení automaticky sbírat informace o provedených SQL příkazech a za běhu bude provádět jejich analýzu pro uložení do vlastní databáze.
3. Žádný SQL příkaz nebude uložený v databázi nástroje více než jednou.
4. Bude implementován také export/import do/z souboru a základní vizualizace databáze.
5. Nástroj bude obsahovat vhodné nastavení; například bude možné přesně specifikovat zachytávané SQL příkazy (t.j. kterých databází/uživatelů/tabulek se mají zachycené SQL příkazy týkat).

V rámci řešení bude provedena rešerše podobných nástrojů, analýza, návrh, implementace a otestování nástroje.

Seznam doporučené odborné literatury:

[1] Yu, Philip S., et al. "On workload characterization of relational database environments." IEEE Transactions on Software Engineering 18.4 (1992): 347-355.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Radim Bača, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 24. dubna 2017 ..... Bauer .....

Rád bych na tomto místě poděkoval svému vedoucímu diplomové práce, panu Ing. Radimu Bačovi, Ph.D., za podmětné připomínky při tvorbě této práce.

Abstrakt

Diplomová práce se zabývá problematikou zachycení typického vytížení SQL databáze. Diplomová práce v první části obsahuje popis problému, použitých technologií, možných řešení problému a porovnání jednotlivých řešení, které by bylo vhodné využít k řešení problému.

V další části diplomové práce je popsán výběr a návrh optimálního řešení, které je implementováno. Po implementaci je začleněno do nástroje pro práci s SQL databází a testováno nad vytížením. Nakonec je zde popis optimalizace a opětovné testování nad vytížením a porovnání výkonu.

Klíčová slova: C#, .NET, SQL, T-SQL, Extended Events, Add-In, SQL Server, Typické vytížení nad SQL Serverem

Abstract

This diploma thesis deals with the capture of a typical SQL database workload. Diploma thesis in the first part contains a description of the problem, a description of the technology, a description of the possible solutions to the problem and comparison of solutions that could be used for solutions.

The next part of this diploma thesis describes the selection and design of optimal solution that is implemented. After the implementation, solution is integrated into tools for working with SQL database and tested over utilization. Finally, there is a description of optimization and re-testing of load and performance comparison.

Key Words: C#, .NET, SQL, T-SQL, Extended Events, Add-In, SQL Server, Typical workload on SQL Server

Obsah

Seznam použitých zkratek a symbolů

SQL	–	Structured Query Language
JMD	–	Jazyk pro manipulaci s daty
JDD	–	Jazyk pro definici dat
XML	–	Extensible Markup Language
GUI	–	Graphical User Interface
API	–	Application Programming Interface
CLR	–	Common Language Runtime
IDE	–	Integrated Development Environment
DLL	–	Dynamic Link Library
FIFO	–	First-in First-out
DTO	–	Data transfer object
DMV	–	Dynamic Management Views

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Často se stává, že v produkčním prostředí je potřeba získat a případně analyzovat SQL příkazy vykonávané v určité SQL databázi. Je to z důvodu toho, že analýza SQL příkazů může být užitečná pro testování, ladění výkonu i získání představy o objemu a typu různých SQL příkazů. Tyto zachycené SQL příkazy tvoří typické vytížení nad SQL databází. Vytížením jsou myšleny jednotlivé SQL příkazy, které se vykonávají v relačním databázovém systému. Těmito SQL příkazy jsou myšleny SQL příkazy ze třídy JMD pro manipulaci s daty a SQL příkaz SELECT pro dotazování. SQL příkazy se často opakují pouze s odlišnými hodnotami parametrů a pro typické vytížení stačí pouze jeden nebo pár vzorků jednoho SQL příkazu. V SQL Server existují nástroje, které umožňují tyto SQL příkazy zaznamenávat. Problémem těchto nástrojů je, že objem takto zaznamenaných dat nemusí být malý a také, že zaznamenaná data vyžadují další zpracování. Také práce s těmito nástroji nemusí být jednoduchá.

Cílem této práce je rešerže těchto existujících nástrojů, jejich porovnání a implementace nástroje, který bude schopen zachytávat typické příkazy nad SQL Server databází a nebude mít výše zmíněné problémy. Tento nástroj bude možné spouštět v rámci SQL Server s minimem oprávnění. Vytvořený nástroj bude dle nastavení schopen automaticky sbírat informace o provedených SQL příkazech a tyto informace analyzovat a ukládat do vlastní SQL databáze, a to tak, že žádný SQL příkaz nebude uložen v SQL databázi více než jednou. Dále bude možné provádět export/import typického vytížení do/z souboru.

Kapitola č. 2 je věnována teoretické části práce, provedení analýzy a popisu nástrojů pro zachytávání typického vytížení nad SQL Server a jejich následného srovnání. V kapitole č. 3 je popis návrhu nástroje pro zachytávání typického vytížení nad SQL Server a jeho funkcionality. V další kapitole č. 4 je proveden návrh a popis optimalizací vytvořeného nástroje. Kapitola č. 5 je věnována experimentům a testování vytvořeného nástroje s využitím vytížení nad SQL databází. V poslední kapitole č. 6 je provedeno shrnutí a závěr celkového řešení této diplomové práce.

2 Teoretická část

První podkapitola této části diplomové práce se zabývá popisem jednotlivých nástrojů, které by bylo možné využít k zachycení prováděných SQL příkazů nad SQL databází. V druhé podkapitole se provede srovnání výhod a nevýhod těchto nástrojů a případné využití jejich možných kombinací.

2.1 Popis nástrojů

2.1.1 Extended Events

Extended Events mají škálovatelnou a konfigurovatelnou architekturu, která umožňuje uživateli sbírat informace z instance SQL Server. V počátcích se s Extended Events pracovalo pouze pomocí jazyka SQL, kde vytvoření, nastavení i provoz Extended Events byl umožněn pouze pomocí skriptů. Od verze SQL Server 2012 existuje v SQL Server Management Studio podpora pro GUI, pomocí něhož lze jednoduše Extended Events nakonfigurovat.

Pomocí Extended Events lze sbírat informace o celé řadě událostí. Událost lze chápat jako libovolnou změnu v SQL databázi (vytvoření, mazání a editace tabulky, vykonání SQL příkazu atd.), ale také i volání SQL dotazu pro získání informací z SQL databáze. Extended Events jsou vysoce škálovatelné. Tím se myslí, že lze nastavit přesně na míru jaké události mají být zachytávané, kdy je možné i vypsat klíčová slova, která se musí nebo nesmí vyskytovat v události, aby byla zaznamenána. Díky tomuto je možné říci, jaké druhy událostí mají být zachyceny, od jakého uživatele mají být zachyceny či z jaké SQL databáze se mají zachytávat.

V rámci Extended Events je možné pro každou událost přidat filtr, pomocí něhož je možné odchytávat a zaznamenávat skutečně pouze informace, které jsou zajímavé. Mimo jiné je tedy možné Extended Events využít pro zachycení SQL příkazů do souboru, který je možné později dále zpracovávat. Události, které lze zachytávat, se dělí do kategorií. V tomto případě je zajímavé zachytávání událostí z kategorie Execution „sp_statement_completed“ a „sql_statement_completed“, které zahrnují všechny SQL příkazy volané nad SQL Server.

Událost „sql_statement_completed“ vznikne vždy, když je SQL příkaz ukončen. To znamená kdykoliv, když je nějaký SQL příkaz dokončen (např. úprava tabulky, vložení záznamu, editace záznamu atd.), vznikne tato událost, a Extended Events provede uložení této události s informací co spustilo tuto událost do souboru.

Událost „sp_statement_completed“ je obdobná jako přechozí událost „sql_statement_completed“, ale s tím rozdílem, že tato událost vznikne, když je dokončen SQL příkaz v rámci uložené procedury [2].

2.1.2 SQL Trace

Je to systém uložených procedur, díky kterým je možné vytvořit tzv. sledování instance SQL Server. Sledování lze chápat jako monitorování aktivity prováděné v rámci SQL Server a její případné ukládání k pozdější analýze. Pomocí sledování lze zachytávat celou řadu událostí prováděných v rámci instance SQL Server. Díky tomu je možné odhalit SQL příkazy či procedury, které ovlivňují výkon SQL Server z důvodu jejich náročnosti na zdroje SQL Server [3].

2.1.2.1 Sledování na straně klienta

Sledování lze vytvořit v SQL Server Profiler, což je GUI pro SQL Trace. Pomocí tohoto GUI lze sledování vytvořit i nastavit podle potřeb. Sledování vytvořené v SQL Server Profiler je možné ukládát do souboru nebo přímo do tabulky. Při spuštění sledování pomocí SQL Server Profiler dochází k vytvoření spojení na instanci SQL Server a poté k sledování událostí. Nevýhodou sledování na straně klienta je nutnost udržovat neustále připojení k SQL Server, což může být někdy obtížné zajistit. Výhodou sledování na straně klienta je, že data získané pomocí tohoto sledování máme u sebe na straně klienta a tyto data můžeme okamžitě analyzovat. Také pro mnoho uživatelů může být výhodou GUI, které při sledování na straně serveru není.

2.1.2.2 Sledování na straně serveru

Sledování lze vytvořit pomocí uložených procedur. Toto sledování se provádí v rámci SQL Server, proto zde není nutné udržovat neustále připojení na SQL Server. Nevýhodou sledování na straně serveru je neexistující GUI, pomocí něhož by se dalo sledování nějakým způsobem nastavit.

2.1.3 Event notifications

Event notifications je mechanizmus pro sledování událostí na úrovni SQL databáze nebo instance SQL Server. Tohle může být provedeno JDD triggery nebo SQL Trace či pomocí Extended Events, ale Event notifications mají tu výhodu, že jsou asynchronní a fungují mimo úroveň transakcí. Informace o událostech jsou odesílány ve formátu XML na SQL Server Service Broker službu. Takže když se vytvoří událost, SQL Server sleduje definované události a posílá informace na SQL Service Broker službu, která je vloží do fronty. Odtud je možné získávat tyto informace asynchronně, když jsou potřeba [4].

2.1.4 Extended Events Reader

Slouží pro čtení dat ze souboru, do kterého zapisuje spuštěná session v instanci SQL Server nebo čtení dat přímo ze streamu běžící session v instanci SQL Server. Ke čtení používá tzv. stream událostí. Je to stream událostí, které přicházejí z běžící session v instanci SQL Server. Stream událostí je asynchronní. Jakmile se stream událostí připojí k instanci SQL Server, perioda odezvy, která se specifikuje při vytváření nového streamu událostí, je snížená na 3 sekundy. Díky tomu dochází ke zpracování událostí téměř v čase vzniku události. Jakmile se stream událostí odpojí, je odezva vrácená na původní hodnotu. Extended Events Reader je možné využít pomocí speciálního API [5].

2.1.5 SQL Server Agent

SQL Server Agent je Microsoft Windows služba, která spouští naplánované administrativní úkoly, které se nazývají joby. Job je posloupnost akcí, které je schopen SQL Server Agent vykonat. Job obsahuje jeden nebo více kroků. Každý krok obsahuje vlastní úkol. SQL Server Agent využívá SQL Server k uložení informací o jobech. U každého jobu je možné říct, v jakých intervalech se má automaticky spouštět [6].

2.1.6 Trigger

Trigger je speciální druh uložené procedury, která se automaticky spouští v reakci na určité akce vyvolané v instanci SQL Server. Existuje hned několik druhů triggeru. JMD trigger je vytvořený nad jednou tabulkou v SQL databázi. Události, které vyvolávají JMD trigger, jsou vložení, editace a mazání záznamů v tabulce. Trigger není možné spouštět explicitně. Jediné možné spuštění triggeru, který je vytvořen pro tabulku, je provedení změny pro danou tabulku. JDD trigger se automaticky spouští v reakci na události SQL příkazů CREATE, ALTER, DROP atd. JDD trigger není vázán na tabulku, ale na SQL databázi [7].

2.1.7 Dynamic Management Views (DMV)

DMV vrací informace o SQL Server nebo SQL databázi, které můžou být využity k monitorování funkčnosti instance SQL Server, diagnostice problémů a k ladění výkonu. Jsou to pohledy a funkce, které jsou ve schématu sys a jejich názvy začínají předponou dm_* [9].

2.1.8 CLR uložené procedury

CLR uložená procedura je speciální databázový objekt v instanci SQL Server, který je naprogramován ve vyšším programovacím jazyce. Manipulace s touto procedurou je stejná jako s normální uloženou procedurou v SQL Server, ale její implementace je uložená v assembly, která je připojená k instanci SQL Server, takže zobrazení těla této uložené procedury není možné. Díky tomu, že CLR uložená procedura je naprogramována ve vyšším programovacím jazyce, je možné v rámci CLR uložené procedury využívat konstrukce těchto jazyků jako jsou pole, generické seznamy, třídy, dědičnost, vlákna atd.

Z důvodu toho, že přidání CLR uložené procedury do SQL Server není úplně běžná záležitost, je nutné prostředí připravit na to, aby bylo možné pracovat s CLR uloženými procedurami v rámci SQL Server. Proto jsou zde vypsány náležitosti, které jsou potřeba udělat, aby bylo možné využívat CLR uložené procedury v SQL Server [8].

CLR integrace je defaultně zakazána a musí se povolit, aby bylo možné využívat objekty, které jsou implementovány pomocí CLR. K povolení je nutné nastavit proměnnou SQL databáze „clr enabled“ na hodnotu „1“, díky čemuž se povolí CLR.

Po povolení CLR je ještě potřeba nastavit „TRUSTWORTHY“ vlastnost. Tato databázová vlastnost říká, zda instance SQL Server věří SQL databázi a jejímu obsahu. Defaultně je tato vlastnost vypnutá. Avšak pro práci s CLR uloženými procedurami je jí potřeba povolit, a to SQL příkazem uvedeným ve výpisu č. 1. Díky tomu je možné připojit assembly.

```
ALTER DATABASE DATABASE_NAME SET TRUSTWORTHY ON
```

Výpis 1: Ukázka SQL příkazu pro nastavení „TRUSTWORTHY“ vlastnosti

Pro vytvoření assembly v instanci SQL Server je nutné vytvořit tuto assembly, a to pomocí SQL příkazu uvedeného ve výpisu č. 2.

```
CREATE ASSEMBLY NAME
FROM ASSEMBLY_PATH
WITH permission_set=UNSAFE
```

Výpis 2: Ukázka SQL příkazu pro vytvoření assembly

Díky tomu bude assembly viditelná v instanci SQL Server a bude možné využívat její funkce jako uložené procedury. Avšak pro vytvoření takovéto assembly je nutné, aby uživatel byl vlastníkem této assembly a také všech assemblies, na které je odkazováno prostřednictvím této assembly. Aby uživatel mohl vytvořit assembly, je nutné, aby měl roli „sysadmin“ a měl právo pro čtení z databáze.

2.1.9 Add-In pro SQL Server Management Studio

Add-In je zkompilovaná DLL knihovna, která je spuštěná v IDE SQL Server Management Studio. Díky tomu je možné vytvořit vlastní rozšíření, které je možné integrovat do nástroje SQL Server Management Studio a toto rozšíření bude spuštěno v rámci procesu „sqlservr.exe“. Jako rozšíření mohou být chápány např. nové položky v menu aplikace, nové položky v kontextovém menu v Object Explorer, úprava editoru atd. [10].

2.1.9.1 Připojení a spuštění Add-In

Při vytváření DLL knihovny pro Add-In je potřeba mít v projektu třídu s názvem „Connect.cs” a v této třídě mít metody „OnConnection” a „Exec”. Tyto metody jsou velmi důležité, protože díky nim je možné připojit a spustit vlastní Add-In v prostředí SQL Server Management Studio.

2.1.9.2 Metoda „OnConnection”

Po spuštění SQL Server Management Studio se kontroluje, zda existují nějaké Add-Ins ve složce „C:\ProgramData\Microsoft\MSEnvShared\Addins”. Jakmile je nalezen XML soubor, který popisuje nějaký Add-In, je z něj získána cesta k DLL knihovně a začne být vykonáván kód metody „OnConnection”. V této metodě se provádí připojení Add-In k prostředí SQL Server Management Studio. Nejprve pomocí příkazu uvedeném ve výpisu kódu č. 3

```
CommandBar menuBarCommandBar = ((Microsoft.VisualStudio.CommandBars.CommandBars
    )_applicationObject.CommandBars)["MenuBar"];
```

Výpis 3: Ukázka C# příkazu pro získání odkazu na „Menu“

se získá odkaz na objekt, který reprezentuje celé horní menu v SQL Server Management Studio. Pomocí tohoto objektu je možné získat odkaz na záložku „Nástroje” pomocí příkazu popsaného ve výpisu kódu č. 4.

```
CommandBarControl toolsControl = menuBarCommandBar.Controls["Tools"];
```

Výpis 4: Ukázka C# příkazu pro získání odkazu na záložku „Nástroje”

Jakmile existuje odkaz na objekt záložky „Nástroje”, stačí nad tímto objektem zavolat metodu s názvem „AddNamedCommand2“ a předat jí parametry jako je název, pod kterým se bude Add-In zobrazovat v záložce „Nástroje”, dále parametr, který říká, že položka bude klikací tlačítko a nebude u ní zobrazena ikona.

2.1.9.3 Metoda „Exec”

Tato metoda se spouští po kliknutí na vytvořenou položku v záložce „Nástroje“. Metoda má vstupní parametr „handled“ datového typu bool, který se musí na začátku metody nastavit na „false“ a po úspěšném vykonání metody se musí nastavit na „true“, aby prostředí SQL Server Management Studio vědělo, že zpracování této metody proběhlo v pořádku. V rámci této metody se provádí zpracování vlastního kódu Add-In. Může to být například zobrazení GUI nebo vykonání libovolné činnosti. Výhodou propojení Add-In s prostředím SQL Server Management Studio je to, že je možné využívat jeho součásti. V tomto případě je využit standardní výstup tohoto prostředí pro výpis informací, aby uživatel věděl co se děje. Do standardního výstupu se zapisuje průběžně během práce s WorkLoad Add-In, který je popsaný v kapitole č. 3.7.

2.1.10 Tabulka s aktuálně zpracovávanými SQL příkazy v SQL Serveru

V SQL Server databázi existuje pohled s názvem „sys.dm_exec_requests“, který obsahuje informace o aktuálních SQL příkazech vykonávaných v rámci SQL Server. Příklad, jak může vypadat SQL dotaz pro získání aktuálně běžících SQL příkazů, je uveden ve výpisu č. 5.

```
SELECT sqltext.TEXT,
req.status,
req.command,
req.total_elapsed_time
FROM sys.dm_exec_requests req
CROSS APPLY sys.dm_exec_sql_text(sql_handle) AS sqltext
```

Výpis 5: Ukázka SQL dotazu pro získání aktuálně běžících SQL příkazů

Tento SQL dotaz vrátí aktuální SQL příkazy vykonávané na databázovém systému. Z funkce „sys.dm_exec_sql_text“, kde vstupním parametrem je „sql_handle“ získaný z pohledu „sys.dm_exec_requests“, je získán text SQL příkazu, který byl zavolán na SQL databázi.

2.1.11 SQL příkazy získané z SQL Server Plan Cache

V SQL Server databázi existuje SQL Server Plan Cache. Při vykonání SQL příkazu se SQL Server nejprve podívá do SQL Server Plan Cache, jestli zde již není plán vykonání pro daný SQL příkaz, a pokud již takový plán existuje, pak jej použije a tím se ušetří čas, který by byl nutný pro vytvoření nového plánu SQL příkazu. Čtení z SQL Server Plan Cache je zobrazeno ve výpisu č. 10.

2.2 Analýza možných řešení

V následujících kapitolách jsou rozebrány možné postupy při zachycení typického vytížení nad SQL Server.

2.2.1 Metoda založená na Extended Events

Kroky řešení:

1. Vytvoření, nastavení a spuštění Extended Events,
2. Ukládání zachycených SQL příkazů pomocí Extended Events do souboru,
3. Vytvoření uložené procedury, která provede čtení z Extended Events souboru a zpracuje všechny SQL příkazy, které nakonec uloží do SQL databáze s typickým vytížením,
4. Vytvoření automatického jobu, který bude spouštět vytvořenou uloženou proceduru z předchozího bodu.

Výhodou Extended Events je to, že je to tzv. odlehčený výkonnostní monitorovací systém, který využívá minimum zdrojů oproti SQL Trace. Další výhodou je možnost zachycení událostí do souboru. Díky široké škálovatelnosti Extended Events je možné nastavit velké množství podmínek, díky kterým se vyfiltrují pouze události, které jsou aktuálně potřeba. Díky této filtraci se částečně sníží velikost výsledného souboru a tím se urychlí následná práce s tímto souborem.

Zde by bylo ještě vhodné uvést možnou optimalizaci, a to využití metody založené na Extended Events v kombinaci s SQL Server Plan Cache. Optimalizace spočívá v tom, že předtím, než je nutné provést čtení ze souboru vytvořeného pomocí Extended Events by se prováděla kontrola, zda v SQL Server Plan Cache přibyly nějaké nové plány od posledního zpracování Extended Events souboru. Pokud by nové plány nepřibyly, pak není nutné načítat Extended Events soubor z disku a provádět analýzu všech SQL příkazů v souboru, což může být časově i výpočetně náročné.

2.2.2 Metoda založená na SQL Trace

Kroky řešení:

1. Vytvoření, nastavení a spuštění SQL Trace pomocí SQL Profiler,
2. Ukládání zachytávaných SQL příkazů do tabulky v SQL databázi nebo souboru,

3. Vytvoření uložené procedury, která provede čtení ze souboru nebo tabulky vytvořené v předchozím bodě a zpracuje všechny SQL příkazy, které nakonec uloží do SQL databáze s typickým vytížením,

4. Vytvoření automatického jobu, který bude spouštět vytvořenou proceduru z předchozího bodu.

Výhodou SQL Trace je jednoduché nastavení pomocí GUI. Jeho nevýhodou je fakt, že server-side sledování není možné ukládát do tabulky. Další nevýhodou SQL Trace je větší vytížení databáze oproti Extended Events [11]. Ale největší nevýhodou se jeví fakt, že Microsoft plánuje odstranit SQL Trace v budoucí verzi SQL Server a již nedoporučuje využívat tuto funkčnost v nových projektech. Náhradou SQL Trace jsou Extended Events. Z tohoto důvodu tohle řešení není vhodné využít.

2.2.3 Metoda založená na Event notifications

Kroky řešení:

1. Vytvoření SQL Server Service Broker služby a její nastavení pro zachycení požadovaných SQL příkazů,

2. Ukládání SQL příkazů pomocí služby do SQL Server Service Broker fronty,

3. Vytvoření uložené procedury, která bude zpracovávat SQL příkazy z fronty a následně je ukládat do SQL databáze s typickým vytížením,

4. Připojení uložené procedury ke frontě, aby se při vložení do fronty mohla uložená procedura vytvořená v předchozím bodě asynchronně spustit.

Výhodou Event notifications je propracované řešení, které je jednoduše nastavitelné. Další výhodou je zpracování událostí asynchronně, když je potřeba. Nevýhodou je, že Event notifications dokáží zpracovávat pouze JDD příkazy a s JMD příkazy či příkazem SELECT si neporadí. Z tohoto důvodu toto řešení není vhodné použít.

2.2.4 Metoda založená na Extended Events Reader

Kroky řešení:

1. Vytvoření Extended Events a ukládání událostí do souboru,

2. Vytvoření externí aplikace, která bude využívat API pro práci s Extended Events Reader,

3. Asynchronní čtení pomocí API z Extended Events souboru a zpracovaní SQL příkazu,

4. Uložení zpracovaného SQL příkazu do SQL databáze s typickým vytížením.

Výhodou Extended Events Reader je zpracování událostí asynchronně. Nevýhodou je nutné neustálé připojení na instanci SQL Server z externí aplikace, která musí obsahovat práci pro získání a zpracování událostí ze streamu událostí. Řešení problému pomocí Extended Events a k němu připojeného readeru není úplně vhodné, protože je zde potřeba externí aplikace, která musí umožňovat neustálé čtení z Extended Events souboru.

2.2.5 Metoda založená na Trigger

Kroky řešení:

1. Vytvoření triggeru s argumentem „INSTEAD OF“, který říká, že tento trigger bude nahrazovat požadovanou operaci (INSERT, UPDATE, DELETE) pro každou existující tabulku v SQL databázi,
2. Při spuštění triggeru se provede získání dat, se kterými se má pracovat a získání SQL příkazu z triggeru,
3. Zpracování SQL příkazu a uložení do SQL databáze s typickým vytížením,
4. SELECT příkazy by bylo nutné zachytávat nějakým jiným způsobem popsaným v této kapitole.

Řešení zachycení událostí pomocí JMD triggeru je velmi nevhodné, jelikož by bylo potřeba vytvořit trigger pro každou tabulku, která je v SQL databázi. To znamená, že pro každou nově vytvořenou tabulku je potřeba vytvořit také trigger, což může mít nemalý dopad na výkon SQL Server. Dále je zde problém, jak zachytávat SELECT příkazy nad databází, protože tyto příkazy JMD trigger zachytávat nedokáže. Jediná výhoda tohoto řešení je jednoduchost implementace, kdy by se vytvořil jeden trigger, který by bylo možné použít jako šablonu pro vytvoření ostatních triggerů.

2.2.6 Metoda založená na aktuálně zpracovávaných SQL příkazech

Tato metoda využívá DMV pohledy a funkce popsané v kapitole č. 2.1.7. Z tohoto důvodu je tato metoda v dalším textu zařazená jako jedna z metod DMV.

Kroky řešení:

1. Vytvoření uložené procedury, která provede získání aktuálně zpracovávaných SQL příkazů a uložení do pomocné tabulky,

2. Každý SQL příkaz v pomocné tabulce se zpracuje a uloží do SQL databáze s typickým vytížením,

3. Vytvoření automatického jobu, který bude spouštět vytvořenou uloženou proceduru v předchozích krocích.

Výhodou využití tabulky s aktuálně zpracovávanými SQL příkazy k získání provedených SQL příkazů na SQL databázi je to, že je možné vyfiltrovat například pouze SQL příkazy, které jsou typu SELECT, INSERT, UPDATE, DELETE. Nevýhodou tohoto přístupu je fakt, že zde není žádná možnost, jak přidat trigger na systémový pohled „sys.dm_exec_requests“. Další nevýhodou je problém s tím, že v aktuální chvíli počet vykonávaných SQL příkazů nebude velký, takže je potřeba spouštět proceduru mnohokrát, než získáme alespoň část typických SQL příkazů.

2.2.7 Metoda založená na SQL Server Plan Cache

Tato metoda využívá DMV pohledy a funkce popsané v kapitole č. 2.1.7. Z tohoto důvodu je tato metoda v dalším textu zařazená jako jedna z metod DMV.

Kroky řešení:

1. Vytvoření uložené procedury, která provede získání SQL příkazů z SQL Plan Cache,

2. Každý SQL příkaz z SQL Plan Cache se zpracuje a uloží do SQL databáze s typickým vytížením,

3. Vytvoření automatického jobu, který bude spouštět vytvořenou uloženou proceduru v předchozích krocích.

Nevýhodou získání SQL příkazů z SQL Server Plan Cache je to, že SQL příkazy jsou již často parametrizované a tím pádem není možné získat hodnoty daných parametrů. Výhodou je získání velkého množství SQL příkazů při jednom čtení SQL Server Plan Cache, protože SQL příkazy setrvávají v SQL Server Plan Cache poměrně dlouhou dobu.

2.2.8 Shrnutí

V této kapitole č. 2.2 bylo vypsáno hned několik možných řešení i s obecným postupem, jak by bylo možné dané řešení implementovat. Taktéž byly u každého možného řešení popsány klady i zápory využití daného řešení. Po srovnání kladů a záporů těchto možných řešení byly vybrány řešení, u kterých klady převažují nad zápory. Jedná se o:

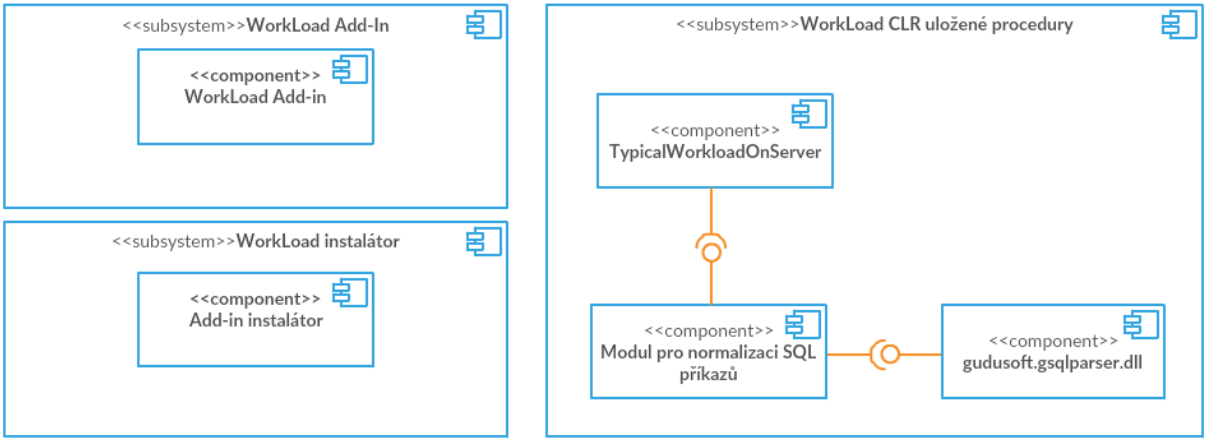
- řešení metodou založenou na Extended Events, které bude popsáno v kapitole č. 3.5,
- řešení metodou založenou na aktuálně zpracovávaných SQL příkazech, které bude popsáno v kapitole č. 3.6.1.1,
- řešení metodou založenou na SQL Server Plan Cache, které bude podrobněji popsáno v kapitole č. 3.6.1.2.

3 Návrh řešení

Tato kapitola se zabývá popisem řešení problému zachycení typického vytížení nad SQL Server. V kapitole č. 3.1 je popsán náhled na celkové řešení, části, ze kterých je celkové řešení vytvořeno, jejich vazby a rozmístění jednotlivých částí celkového řešení. V kapitole č. 3.3 je popsán návrh a využití CLR uložených procedur. V další kapitole č. 3.4 je uveden popis, jak probíhá zpracování zachycených SQL příkazů. V kapitolách č. 3.5 a 3.6 je uveden detailní popis řešení pomocí metod DMV a Extended Events. Nakonec v poslední kapitole č. 3.7 je popsáno, jak se pracuje s metodami DMV a Extended Events v rámci WorkLoad Add-In, a jaké další možnosti práce nabízí tento Add-In.

3.1 Náhled celkového řešení

Na obr. č. 1 je zobrazen diagram komponent, který přehledněji popisuje, jak je celkové řešení propojeno mezi sebou a z jakých částí se skládá. Jak lze vidět v diagramu komponent, celé řešení se skládá ze tří větších subsystémů.

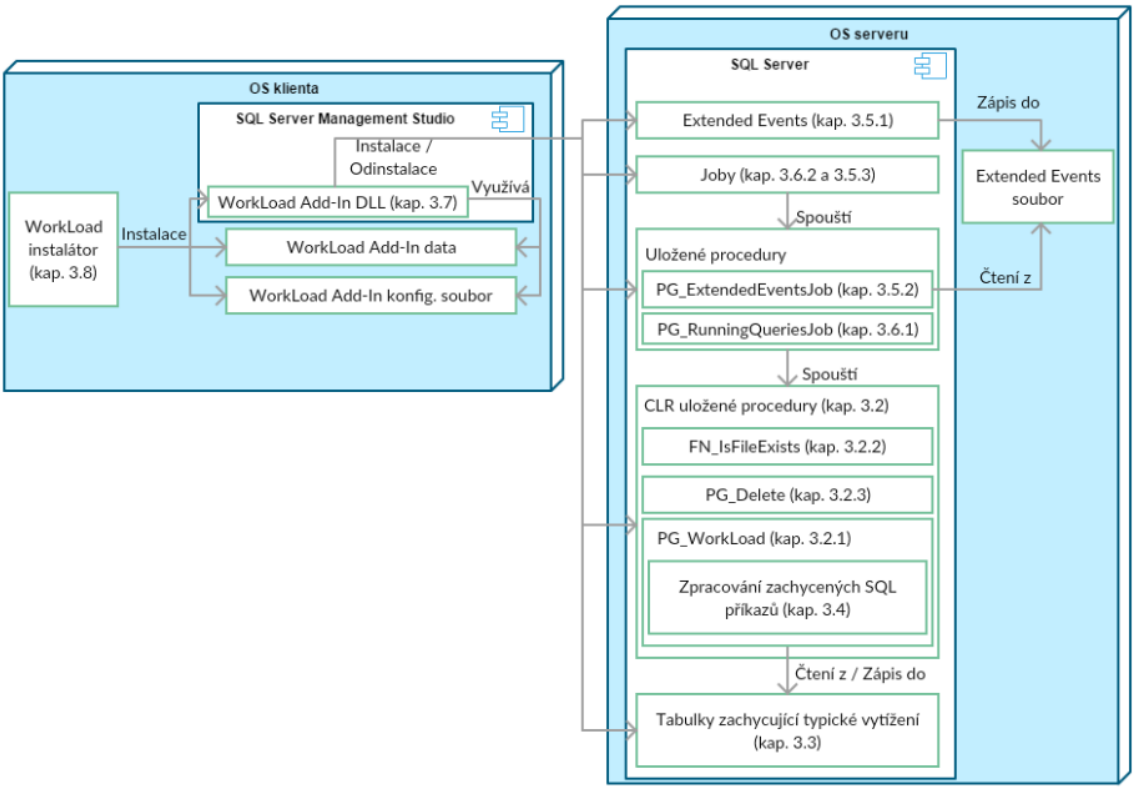


Obrázek 1: Diagram komponent

WorkLoad Add-In subsystém je integrován do SQL Server Management Studio jako rozšíření. Integrace s SQL Server Management Studio umožňuje využití funkcí jeho prostředí a tím zjednodušení implementace.

WorkLoad CLR uložené procedury subsystém slouží pro zpracování zachycených SQL příkazů, jejich normalizaci a uložení do SQL databáze, kde se ukládají zpracované SQL příkazy.

WorkLoad instalátor subsystém slouží pro instalaci a odinstalaci celého řešení na hostitelském počítači prostřednictvím GUI. Díky tomu je instalace celého řešení pro uživatele značně zjednodušena.



Obrázek 2: Náhled rozmístění jednotlivých částí celkového řešení a jejich vazby

Na obrázku č. 2 je možné vidět umístění jednotlivých součástí celkového řešení v rámci počítačů a také vzájemné vazby mezi těmito částmi.

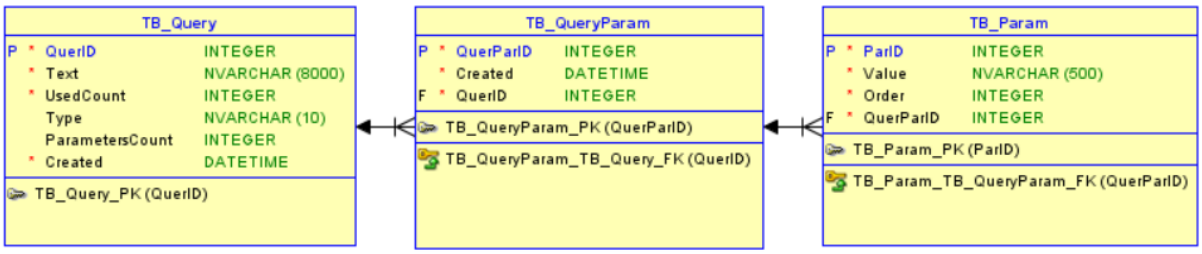
První část, která obsahuje WorkLoad instalátor, WorkLoad Add-In DLL, WorkLoad Add-In data a WorkLoad Add-In konfigurační soubor, je umístěná na klientském počítači, přičemž samotný WorkLoad Add-In je v rámci SQL Server Management Studio jako Add-In.

Druhá část je umístěná na počítači, kde poběží samotný SQL Server. Tato část obsahuje Extended Events, joby, uložené procedury, CLR uložené procedury a tabulky zachycující typické vytížení, jež jsou umístěny v rámci SQL Server databáze a Extended Events soubor, který je umístěn mimo SQL Server.

3.2 Tabulky zachycující typické vytížení

Pro uložení zachycených SQL příkazů reprezentujících typické vytížení nad SQL databází je potřeba vytvořit v SQL databázi tabulky tomu určené. Pro tyto účely je potřeba v SQL databázi vytvořit tři tabulky a mezi nimi odpovídající relace jak lze vidět na obr. č. 3. Tyto tabulky s relacemi představují typické vytížení nad SQL databází a společný způsob uložení vytížení pro všechny implementované techniky odchycení SQL příkazů. Tabulka „TB_Query“ slouží pro

uložení jedinečných SQL příkazů a k nim údajů jako je text SQL příkazu, počet výskytů, typ o jaký SQL příkaz se jedná, počet parametrů a datum s časem vytvoření. Tabulka „TB_Param“ reprezentuje parametry SQL příkazu, u kterých se eviduje hodnota a pořadí parametru v SQL příkazu. Poslední tabulka „TB_QueryParam“ reprezentuje jednotlivé zpracování SQL příkazu, u kterého se eviduje datum a čas vytvoření. Díky takto vytvořeným relacím je možné získat pro každý SQL příkaz jeho parametry v rámci jednoho zpracování.



Obrázek 3: Schéma tabulek a jejich relací v SQL databázi

3.3 CLR uložené procedury a funkce

Tato kapitola se zabývá analýzou SQL příkazu, jeho parametrizací a normalizací a uložením do SQL databáze, která obsahuje typické vytížení. Jak už název napovídá, jedná se o CLR uložené procedury, které jsou ve skutečnosti statické funkce psány v tomto případě v jazyce C#. Tyto funkce obsahují speciální anotaci, která udává o jaký SQL objekt (uložená procedura, funkce, trigger atd.) se jedná. V tomto případě se jedná o uložené procedury. CLR uložené procedury bylo potřeba vytvořit dvě, a to „PG_WorkLoad“, která je popsána v kapitole č. 3.3.1 a „PG_DeleteFile“, která je popsána v kapitole č. 3.3.3. CLR funkci bylo potřeba vytvořit pouze jednu, a to „FN_IsFileExists“, která je popsána v kapitole š. 3.3.2. Na obr. č. 2 lze vidět umístění a také vazby CLR uložených procedur v rámci celého řešení. Bližší využití jednotlivých CLR uložených procedur a funkcí je zobrazeno na obr. č. 8 a 10.

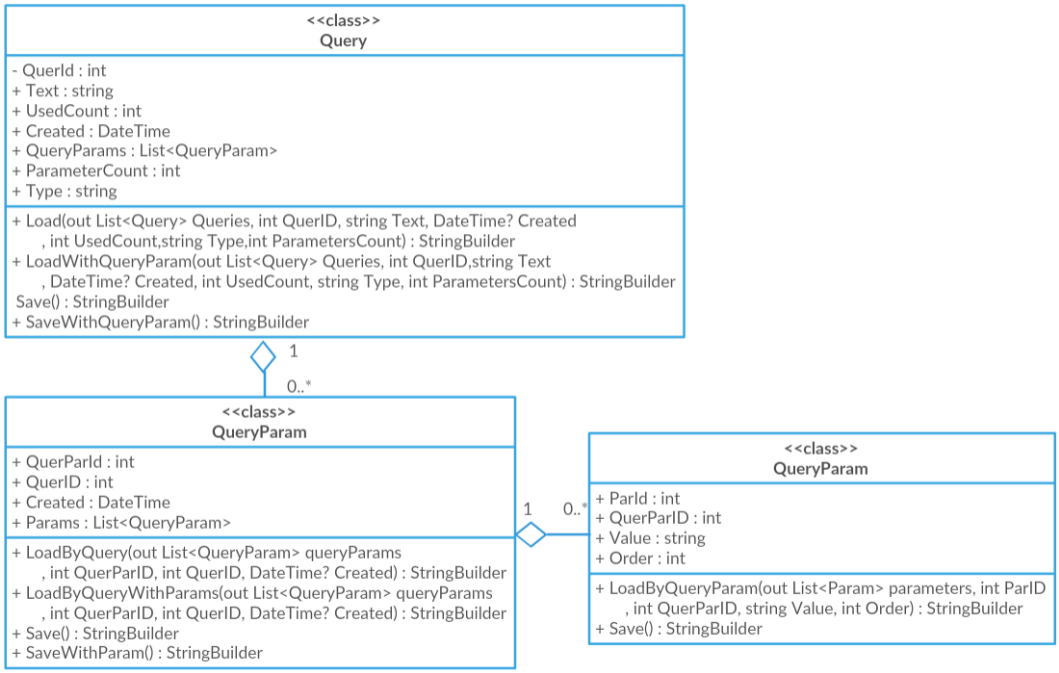
3.3.1 CLR uložená procedura „PG_WorkLoad“

Pomocí této CLR procedury se upraví SQL příkaz a uloží se do tabulek zachycujících typické vytížení popsané v kapitole č. 3.2. Vstupní parametr CLR uložené procedury je SQL příkaz a jako výstupní parametr je chybová hláška, pokud se vyskytne nějaká chyba během zpracování. Využití této CLR uložené procedury je zobrazeno na obr. č. 8 a 10.

V první fázi dochází ke zpracování SQL příkazu pomocí modulu pro normalizaci SQL příkazů, který je popsán v kapitole 3.4. Výstupem první fáze je text SQL příkazu, který je již parame-

trizován a normalizován, dále je rozpoznáno o jaký typ SQL příkazu se jedná. Taktéž je znám počet parametrů zpracovávaného SQL příkazu, jejich pořadí a hodnoty těchto parametrů.

V druhé fázi po zpracování SQL příkazu se provádí naplnění daty získanými jako výstup z první fáze zpracování SQL příkazu objekty tříd Query, QueryParam a Param. Tyto třídy jsou namapovány do SQL databáze pomocí vztahu 1:1. Schéma tabulek v SQL databázi je zobrazeno na obr. č. 3.

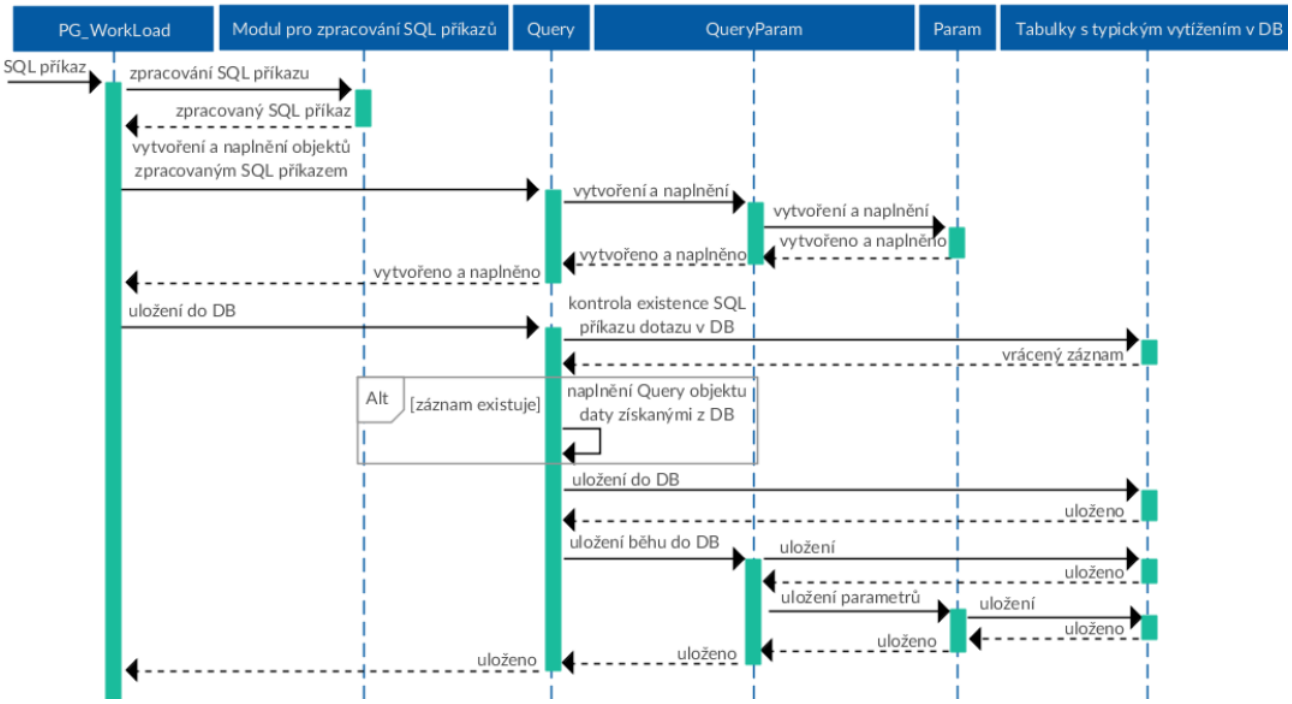


Obrázek 4: Třídní diagram DTO mapování

Mapování 1:1 znamená, že 1 třída bude 1 tabulka v SQL databázi. Takže bylo nutné vytvořit tři třídy, a to „Query“, „QueryParam“ a „Param“. Jak jsou tyto třídy vytvořeny a propojeny lze vidět v třídním diagramu na obr. č. 4. Dále je potřeba SQL příkaz uložit do SQL databáze, která obsahuje typické vytížení SQL Server. Popis tabulek zachycujících typické vytížení je v kapitole č. 3.2.

Před tím, než se uloží SQL příkaz do tabulek zachycujících typické vytížení v SQL databázi, se provede kontrola, jestli už takový SQL příkaz neexistuje. Kontrola se provede tak, že se zavolá metoda „Load“ z třídy „Query“ s parametrem „Text“, do kterého se vloží text zpracovávaného SQL příkazu. Uvnitř metody se provede volání SQL dotazu do SQL databáze s podmínkou na atribut „Text“. Pokud metoda vrátí objekt třídy „Query“, pak již SQL příkaz existuje. Pokud SQL příkaz existuje, stačí zvýšit počítadlo „UsedCount“ o 1 a nakonec zavolat metodu „SaveWithQueryParam“ z třídy „Query“, která uloží aktualizovaný SQL příkaz a následně objekty třídy

„QueryParam“ a objekty třídy „Param“. Jakmile skončí ukládání do SQL databáze, a pokud nenastala žádná chyba, dojde k ukončení vykonání CLR uložené procedury a výstupní parametr „error“ bude prázdný. Diagram vyjadřující popsanou logiku můžeme vidět na obr. č. 5.



Obrázek 5: Průběh zpracování v CLR uložené proceduře „PG_WorkLoad“

3.3.2 CLR funkce „FN_IsFileExists“

Slouží k zjištění, zda soubor existuje. Vstupním parametrem je cesta k souboru. Pokud soubor existuje, pak návratová hodnota CLR funkce je „true“, a pokud neexistuje, pak je „false“. Kontrola na existenci souboru byla vytvořena jako CLR funkce z toho důvodu, že je využívaná v proceduře k ověření existence souboru. Kdyby bylo ověřování implementováno v jazyce T-SQL, pak by uživatel musel mít další práva, což je nežádoucí, protože snahou je využití co nejmenšího množství práv. Volání této funkce je součástí logiky popsané na obr. č. 8.

3.3.3 CLR uložená procedura „PG_DeleteFile“

Slouží k vymazání souboru v souborovém systému. Parametrem je cesta k souboru nebo složce, kterou chceme vymazat. Z cesty zadané parametrem lze rozpoznat, zda se jedná o cestu ke složce nebo k souboru. Podle toho je možno získat pole cest k vymazání, pokud se jedná o složku, anebo jednu cestu, jestliže se jedná o soubor. Poté se projde pole cest a provede se vymazání. Zde je

stejný problém jako u „FN_IsFileExists“. Kdyby bylo implementováno vymazání souboru v rámci T-SQL, pak by byla po uživateli vyžadována další práva, což není cílem práce. Volání této uložené procedury je součástí logiky popsané na obr. č. 8.

3.4 Zpracování zachycených SQL příkazů

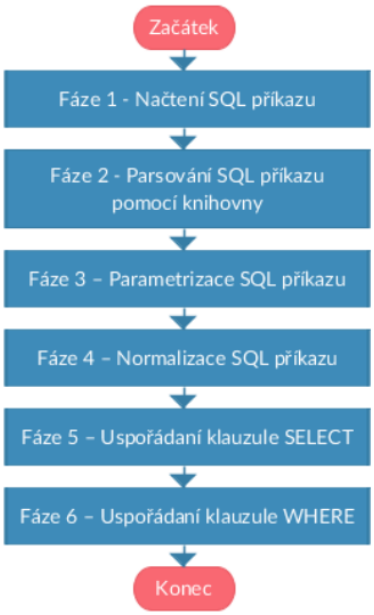
Pro zpracování zachycených SQL příkazů byl použit „modul pro normalizaci SQL příkazů“, který byl vytvořen v rámci školního semestrálního projektu pod vedením pana Ing. Radima Bači, Ph.D. (Tomáš Bauer, 2016). Hlavní funkcí modulu je provedení normalizace SQL příkazu. Normalizací SQL příkazu rozumíme proces, při kterém dojde k transformaci SQL příkazu do normalizované formy, ve které budou dva SQL příkazy shodné jen v případě, že mají stejnou sémantiku. Odstraňují se takto rozdíly mezi SQL příkazy, které vznikly různou syntaxí stejné sémantiky. Příklad dvou SQL dotazů, které mají různou syntaxi, ale stejnou sémantiku, lze vidět na obrázku č. 6.

```
SELECT P.FIRST_NAME, P.LAST_NAME, P.YEAR_OF_BIRTHDAY, P.STREET, P.CITY
FROM TB_Person P
WHERE (P.CITY = 'Ostrava' OR P.CITY= 'Brno') AND P.YEAR_OF_BIRTHDAY > 1990

SELECT O.YEAR_OF_BIRTHDAY, O.STREET, O.FIRST_NAME, O.CITY, O.LAST_NAME
FROM TB_Person O
WHERE O.YEAR_OF_BIRTHDAY > 1990 AND (O.CITY = 'Ostrava' OR O.CITY= 'Brno')
```

Výpis 6: Ukázka dvou SQL dotazů, které mají stejnou sémantiku

Zpracování zachycených SQL příkazů se provádí v jednotlivých fázích. Názvy jednotlivých fází a jejich posloupnost je zobrazena na obr. č. 6.



Obrázek 6: Diagram popisující chod modulu

Fáze 1 – Načtení SQL příkazu
Načtení SQL příkazu probíhá v rámci CLR uložené procedury „PG_WorkLoad“ popsané v kapitole č. 3.3.1, kde je SQL příkaz předáván jako vstupní parameter. Následně je SQL příkaz předán modulu pro normalizaci SQL příkazu. Po předání SQL příkazu je načtení u konce a přechází se do fáze č. 2.

Fáze 2 – Parsování pomocí knihovny
Jakmile je SQL příkaz načten, provede se parsování SQL příkazu pomocí knihovny „gudusoft.gsqlparser.dll“. Tímto se z SQL příkazu stane objekt. Pokud se nepovede SQL příkaz rozparsovat, vypíše se chybová hláška a zpracování se ukončí. Pokud se povedlo SQL příkaz rozparsovat, přejde do fáze č. 3.

Fáze 3 – Parametrizace SQL příkazu
V této fázi dochází k průchodu SQL příkazu po jednotlivých tokenech, což jsou nejmenší nedělitelné jednotky. Během průchodu se rozlišuje několik tzv. módů. Podle toho, v kterém módu program je, tak pracuje. Během průchodu, když se provede nahrazení konstanty za proměnnou, se tato konstanta s danou proměnnou uloží do seznamu, aby bylo možné s tímto seznamem dále pracovat. Po průchodu všemi tokeny je parametrizace SQL příkazu hotová.

Pro parametrizaci SQL příkazu bylo potřeba zjistit případy, kdy je či není možné provádět

nahrazování konstant za parametry. Po průzkumu bylo zjištěno, že není možné provádět parametrizaci konstant u datových typů, u kterých se definuje např. rozsah datového typu. Například u datového typu CHAR(size), který se může vyskytovat například ve funkci CONVERT, není možné provést nahrazení konstanty za parametr.

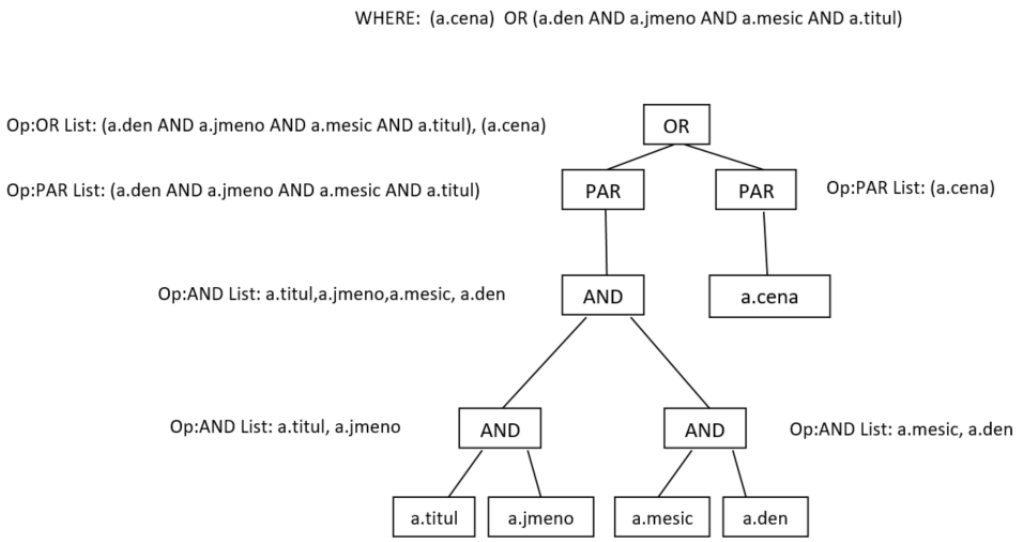
Módy:
IN_MODE – mód, do kterého se program dostane po přečtení klíčového slova IN. Pokud je program v tomto módu, vše co je za tímto klíčovým slovem se bude nahrazovat jedinou proměnnou až do doby, kdy se přečte ukončovací závorka.
NORMAL_MODE – mód, ve kterém se provádí okamžité nahrazování konstanty proměnnou.
BLACK_LIST_MODE – mód, do kterého se program dostane tehdy, když přečte nějaké slovo ze seznamu označeného jako black list. Jsou to datové typy (CHAR, VARCHAR atd.) u kterých se nesmí nahrazovat konstanty za proměnné.
TOP_MODE – mód, do kterého se program dostane přečtením klíčového slova TOP. V tomto módu se zjišťuje, zda konstanta za klíčovým slovem TOP je v závorkách, nebo není (z důvodu toho, že za klíčovým slovem TOP musí být proměnná v závorkách). Pokud konstanta nebyla v závorkách, pak se za klíčové slovo TOP vloží závorky a do nich se vloží nahrazující proměnná. Pokud konstanta byla v závorkách, provede se pouze nahrazení proměnné konstantou.

Fáze 4 – Normalizace SQL příkazu
V této fázi dochází ke generování a přiřazení nových aliasů tabulkám a jejich atributům použitým v SQL příkazu. Nejprve je nutné získat seznam tabulek použitých v SQL příkazu a pro ně získat z databáze jejich atributy. Jakmile je hotov seznam tabulek a jejich atributů, stačí vytvořit metodu, která pro vstupní parametry „název tabulky" a „název sloupce" určí k jaké tabulce daný atribut patří, a tato metoda se zavolá během provádění parsování do objektu. Nakonec stačí získat seznam tabulek, které jsou v SQL příkazu, zaměnit u nich alias za nový vygenerovaný alias, projít jeho seznam připojených sloupců (seznam výskytů sloupců v SQL příkazu) a u každého sloupce přidat alias dané tabulky.

Fáze 5 – Uspořádaní klauzule SELECT
V této fázi se provádí uspořádaní klauzule SELECT. Vytvoří se seznam atributů, které jsou v klauzuli SELECT a poté se tento seznam setřídí podle abecedy. Nakonec se přepíše původní klauzule SELECT na novou uspořádanou klauzuli SELECT.

Fáze 6 – Uspořádaní klauzule WHERE
Klauzule WHERE obsahuje podmínky selekce záznamů. Pokud se zaměříme na samostatnou strukturu této klauzule, je možné zjistit, že důležité operátory mezi jednotlivými podmínkami jsou AND a OR. Také se nesmí zapomenout na závorky. Díky tomuto je možné vytvořit strom, a to tak, že tyto tři zmíněné operátory tvoří uzly stromu a listy stromu tvoří jednotlivé samostatné podmínky. Ukázka vytvořeného stromu je zobrazena na obr. č. 7.

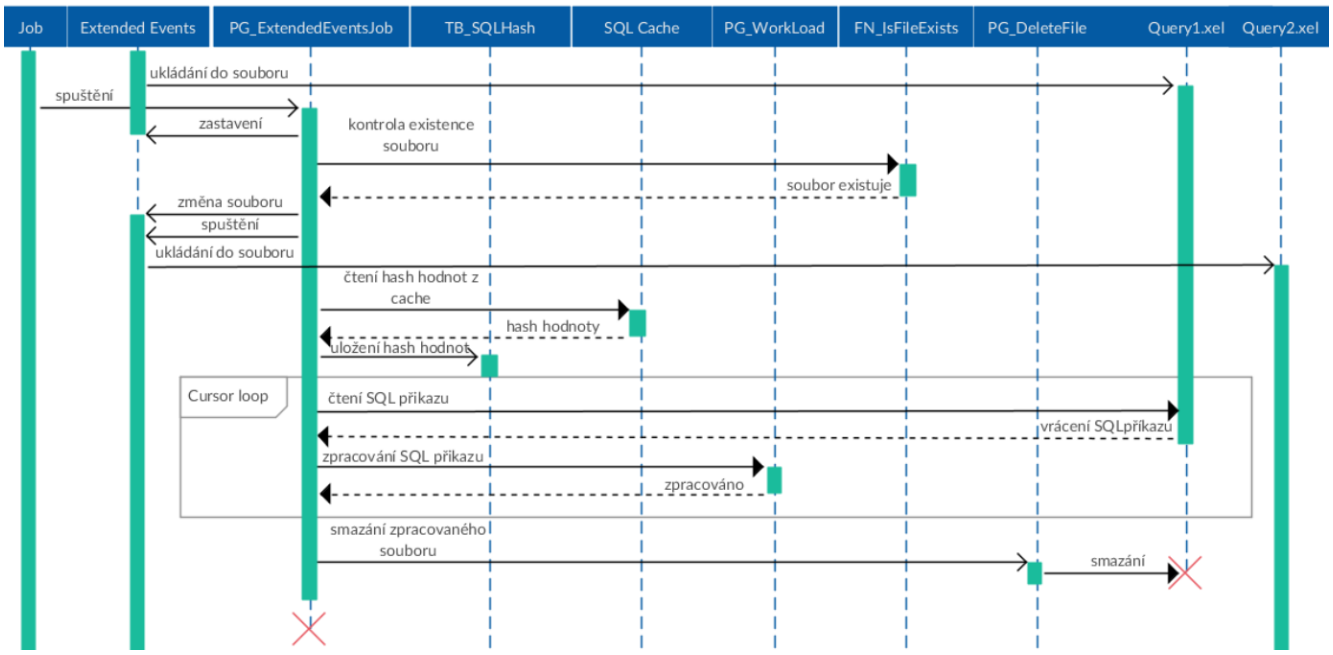
V této fázi se provádí uspořádaní podmínek v klauzuli WHERE. Nejprve se vytvoří strom pro klauzuli WHERE a poté se metodou průchodu Post-order začne tento strom procházet. Dokud se operátor nezmění, stále se přidávají do seznamu jednotlivé uzly a je nutné si pamatovat, jaký je mezi nimi operátor. Jakmile se operátor změní, provede se uspořádání seznamu uzlů. Po dokončení uspořádaní se provede sjednocení těchto uzlů do jediného, a to tak, že se mezi ně přidá operátor. Jakmile je seskupení hotovo, vytvoří se nový seznam a do tohoto seznamu se vloží tento nový uzel. Opět se nastaví operátor pro tento seznam uzlů a s přidáváním se pokračuje dokud opět nedojde ke změně operátoru nebo není konec. Jakmile se dojde ke kořenu, stačí provést sjednocení a výsledná klauzule WHERE je hotová.



Obrázek 7: Stromová struktura klauzule WHERE

3.5 Metoda založená na Extended Events

Toto řešení využívá hned několik technik vhodně kombinovaných tak, aby pomocí nich šlo zachytit SQL příkazy, které jsou prováděné nad SQL databází, a postupně tyto SQL příkazy zpracovat tak, aby výsledkem bylo typické vytížení nad SQL Server uložené v tabulkách popsaných v kapitole č. 3.2. Jádrem řešení je kurzor, který prochází SQL příkazy z Extended Events souboru, a tyto SQL příkazy jsou zpracovány. Jak vypadá získání a zpracování SQL příkazů metodou založenou na Extended Events lze vidět na obr. č. 8.



Obrázek 8: Průběh zpracování metody založené na Extended Events

3.5.1 Extended Events

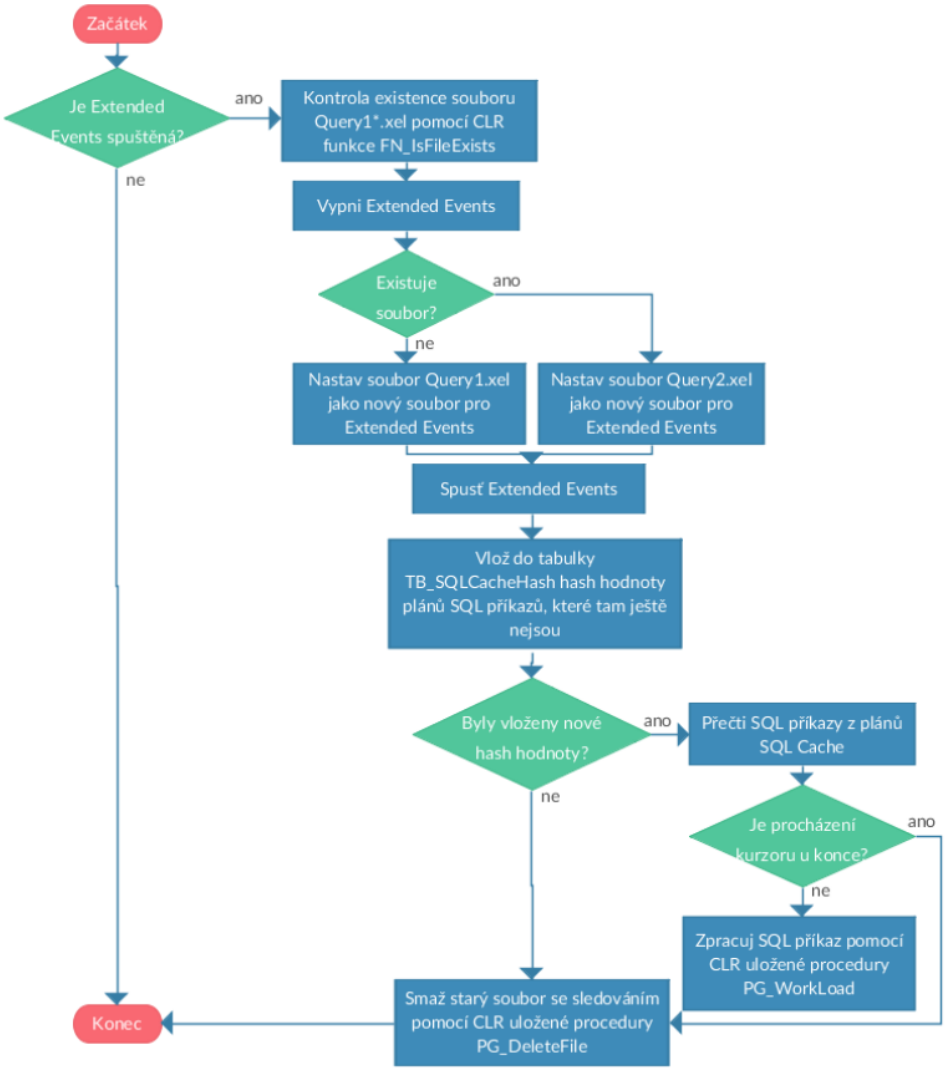
Jak již bylo zmíněno v teoretické části v kapitole č. 2.1.1, Extended Events dokáží zachytávat různé události nad SQL databází a ukládat tyto události hned do několika výstupů. Tímto výstupem může být soubor se všemi událostmi, který je uložen na disku s příponou „.xel“. Cestu k souboru si může uživatel vybrat sám. Název souboru je předem specifikován a nesmí se měnit. Ale i přesto se do názvu souboru, který bude uložen na disku, vkládá jako postfix časové razítko ve formě long int čísla. Další možností je kruhový buffer, který neukládá události na disk, ale ponechává si je v paměti v datové struktuře typu FIFO. V tomto případě byl vybrán pro ukládání dat soubor, a to z toho důvodu, že se k datům dostavám v předem nespecifikovaném intervalu, takže je potřeba, aby data byla všechna, nezměněna a nepřepsána novými

daty. Soubor, do kterého se budou události ukládat, se po vytvoření Extended Events jmenuje „Query1_postfix.xel".

Aby nebylo potřeba zachytávat všechny SQL příkazy, které byly provedeny, byla přidána filtrace na název SQL databáze, pro kterou se mají zachytávat příkazy a dále filtrace na SQL příkazy INSERT, UPDATE, DELETE a SELECT. Nakonec byla přidána filtrace na SQL příkazy, které obsahují klíčová slova, jako jsou názvy tabulek, uložených procedur a funkcí, které jsou volány během zpracování. Tato filtrace je důležitá, protože by se potom zbytečně zpracovávaly neustále stejné SQL příkazy, které nechceme ve výsledném typickém vytížení. Po spuštění metody založené na Extended Events, může uživatel měnit některá nastavení chodu Extended Events. Avšak název Extended Events musí být zachován, aby s ním bylo možné manipulovat v rámci WorkLoad Add-In. Ukázka skriptu pro vytvoření Extended Events je zobrazena ve výpisu č. 14, který je umístěn v přílohách.

3.5.2 Uložená procedura „PG_ExtendedEventsJob"

Je to uložená procedura, která zpracovává události ze souboru zachycené pomocí Extended Events. Tato uložená procedura se spouští v pravidelných intervalech pomocí jobu. Uložená procedura „PG_ExtendedEventsJob" má pouze jeden parametr, a tím je příznak, jestli se při spuštění mají vypisovat ladicí informace. Při nasazení se spouští bez jakýchkoliv parametrů. Využití této uložené procedury je zobrazeno na obr. č. 8. Detailnější pohled této uložené procedury je na obr. č. 9.



Obrázek 9: Detail vykonání uložené procedury „PG_ExtendedEventsJob“

Po spuštění „PG_ExtendedEventsJob“ se provádí kontola, jestli je Extended Events spuštěná, a pokud není, vykonání uložené procedury ukončí. Kontrola se provádí, protože pokud není ExtendedEvents session spuštěná, pak nebude existovat žádný soubor s událostmi, a tudíž je zbytečné, aby se provádělo vyhledávání souboru na disku. Jestliže je spuštěná, provede se kontrola, zda existuje soubor „Query1*.xel“ v cestě, kterou si uživatel sám vybral pomocí CLR funkce „FN_IsFileExists“. Pokud soubor existuje, pak se uloží tato cesta jako aktuální cesta k souboru Extended Events. Pokud soubor neexistuje, pak je jasné, že soubor do kterého se aktuálně ukládají události, bude mít název „Query2*.xel“. Díky tomu, že na začátku byla provedená kontrola na to, jestli je Extended Events session spuštěná, musí jeden z těchto dvou souborů existovat. Oba tyto názvy souborů jsou dány natrvalo a není možné tyto názvy měnit, protože by poté nefungovalo načítání ze souboru z důvodu jeho nenalezení v požadované cestě na disku. Jakmile je jasné, který soubor je aktuální, dojde ke zrušení ukládání událostí do tohoto souboru. Poté

se nastaví ukládání do nového souboru, podle toho, který soubor není nastaven jako aktivní. K výměně souboru je nutné zastavit Extended Events, provést výměnu souboru a opět spustit. Jak tato výměna souborů probíhá je znázorněno ve výpisu kódu č. 7.

```
--Disable logging to file
SET @Sql='alter EVENT SESSION '+@EventName+' on server'+@crlf+
      'drop target package0.event_file'
EXEC sp_executesql @Sql;

--Configure new logging file
select @filePath=REPLACE(filepath,'*','') from @filePathTable where actual=0

SET @Sql='alter EVENT SESSION '+@EventName+' on server'+@crlf+
      ' ADD TARGET package0.event_file(SET filename='''+@filePath+''')'
EXEC sp_executesql @Sql;
```

Výpis 7: Ukázka skriptu pro výměnu souborů pro uložení událostí

Když už je nastaven nový soubor pro ukládání událostí, provede se čtení hash hodnot jednotlivých plánů z SQL Server Plan Cache, ve kterém jsou obsaženy plány vykonání SQL příkazů a následné uložení hash hodnot těchto plánů do tabulky „TB_SqlCacheHash“. Uloží se vždy jen hash hodnoty těch plánů, které ještě nejsou v této tabulce. Hash hodnoty se do tabulky ukládají z důvodu optimalizace, protože pokud se nenajde v SQL Server Plan Cache nový plán, jehož hash hodnota ještě není v tabulce, pak není důvod číst soubor s událostmi a provádět analýzu. Pokud se tedy nevloží nová hash hodnota do tabulky „TB_SqlCacheHash“, zpracování je u konce a soubor, ze kterého by se četlo v případě vložení nových záznamů, se může smazat pomocí CLR uložené procedury „PG_DeleteFile“. Vložení SQL příkazů z SQL Server Plan Cache je vidět ve výpisu kódu č. 8. Detailní vykonání této uložené procedury je znázorněno na obr. č. 9.

```
insert into TB_SqlCacheHash
SELECT DISTINCT ss.query_plan_hash
FROM ( SELECT DISTINCT plan_handle
   FROM sys.dm_exec_cached_plans WITH(NOLOCK)) AS qs
OUTER APPLY sys.dm_exec_query_plan(qs.plan_handle) tp
INNER JOIN sys.databases d ON tp.dbid = d.database_id
inner join sys.dm_exec_query_stats ss on ss.plan_handle=qs.plan_handle
where d.name='##DATABASE_NAME##' and not exists(select null from
    TB_SqlCacheHash sch where sch.plan_hash=ss.query_plan_hash)
```

Výpis 8: Ukázka skriptu pro vložení hash hodnot do tabulky „TB_SqlCacheHash“

Pokud je potřeba číst SQL příkazy z Extended Events souboru, využívá se k tomu funkce „sys.fn_xe_file_target_read_file“, kde jako parametr je cesta k souboru, který je potřeba číst. Data jsou reprezentována v SQL dotazu pomocí atributu „event_data“, jak lze vidět ve výpisu kódu č. 3.5.3. V „event_data“ jsou záznamy uloženy pomocí XML. Ve výpisu kódu č. 9 je možné vidět strukturu XML pro jeden záznam v Extended Events souboru. Nejdůležitější je uzel s názvem „statement“, ve kterém je uložen text SQL příkazu. Z důvodu optimalizace nebudou převáděna „event_data“ na datový typ XML a pak se v něm nebude hledat pomocí XPath. Bude ale provedeno vyhledání textu SQL příkazu pomocí funkcí T-SQL „SUBSTRING“ a „CHARINDEX“. Podrobný popis této optimalizace je uveden v kapitole č. 4.

```
<event name="sql_statement_completed" package="sqlserver" timestamp="2017-04-21
    T12:30:06.144Z">
  <data name="duration"> <value>98</value> </data>
  <data name="cpu_time"> <value>0</value> </data>
  <data name="physical_reads"> <value>0</value> </data>
  <data name="logical_reads"> <value>3</value> </data>
  <data name="writes"> <value>0</value> </data>
  <data name="row_count"> <value>1</value> </data>
  <data name="last_row_count"> <value>1</value> </data>
  <data name="line_number"> <value>1</value> </data>
  <data name="offset_end"> <value>112</value> </data>
  <data name="statement">
    <value>
      <![CDATA[select jmeno,popis,vytvoreni,konec from aukce where id=10]]>
    </value>
  </data>
  <action name="is_system" package="sqlserver"> <value>false</value> </action>
</event>
```

Výpis 9: Ukázka struktury záznamu v Extended Events souboru

Načtené SQL příkazy získané z Extended Events souboru je potřeba projít pomocí kurzoru a pro každý získaný SQL příkaz se spustí CLR uložená procedura s názvem „PG_WorkLoad“, kde jako vstupní parametr bude text SQL příkazu a výstupním parametrem bude chybová hláška, pokud nastane chyba během zpracování. Po skončení průchodu pomocí kurzoru je zpracování u konce. Nakonec se provede vymazání procházeného souboru pomocí CLR uložené procedury „PG_DeleteFile“. Ve výpisu kódu č. 16 je vidět jak se provádí parsování SQL příkazů z Extended Events souboru a následné zpracování těchto SQL příkazů pomocí CLR uložené procedury „PG_WorkLoad“ popsané v kapitole č. 3.3.1.

3.5.3 Automatické spouštění procedur s pomocí Job

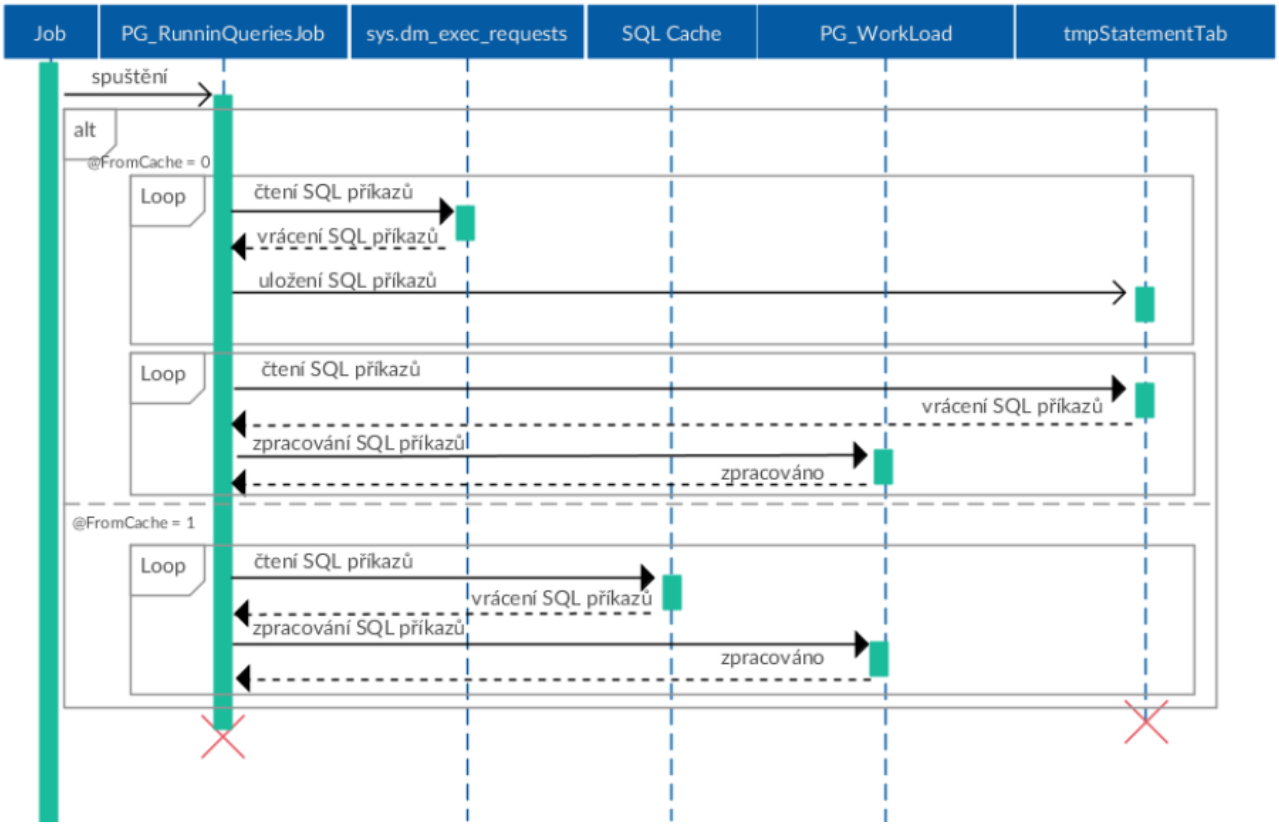
Pro spuštění uložené procedury „PG_ExtendedEventsJob“ popsané v kapitole č. 3.5.2 v pravidelných intervalech je vhodné využít SQL Agent a job. Díky tomu byl vytvořen vlastní job, který má uživatelsky nastavitelné jméno při vytváření jobu. Tento i kterýkoliv jiný job musí mít ke správnému běhu vytvořené tři části:

- **Vytvoření jobu** - job se vytváří pomocí uložené procedury „sp_add_job“. Jedním z parametrů této procedury je i název jobu, který si volí uživatel, a kategorie jobu, která je defaultně nastavená jako „Data Collector“. Dále je možné přidat popis jobu, vlastníka jobu atd. Díky pojmenování jobu je možné jej rozeznat od ostatních jobů, které jsou vytvořeny v SQL Agent. Využití tohoto jobu je zobrazeno na obr. č. 8.
- **Krok jobu** - krok jobu se vytváří pomocí uložené procedury „sp_add_jobstep“. Jedním z parametrů je „ID“ jobu, který byl vytvořen v předchozí části. V kroku se definuje, co se bude dít během spuštění jobu. V tomto případě se spustí uložená procedura „PG_ExtendedEventsJob“. Dále je možné u kroku nastavit, co se má stát, když se vyskytne chyba během zpracování daného kroku.
- **Plán spuštění** - plán spuštění se vytváří pomocí uložené procedury „sp_add_jobschedule“. Jedním z parametrů je „ID“ jobu, ke kterému se má plán připojit. Tento plán slouží jako časovač. Díky němu se job spouští v intervalu, který je u plánu nastaven. V tomto případě se v plánu nastavilo, aby se job spouštěl každou minutu.

Kterákoliv z popisovaných částí jobu může být kdykoliv po prvotním vytvoření změněná, avšak název jobu musí být zachován. Je to z toho důvodu, že kdyby bylo potřeba job odstranit pomocí WorkLoad Add-In, pak by již nebylo možné dohledat tento job a nebyl by odstraněn [1].

3.6 Metody DMV

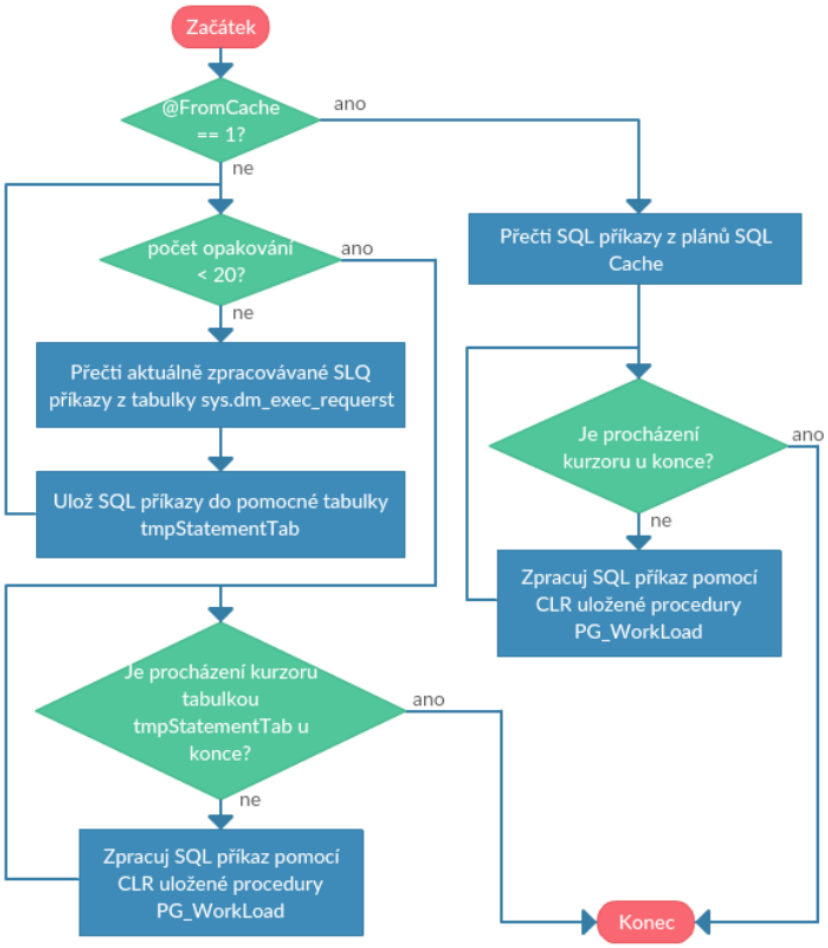
Ve srovnání s metodou založenou na Extended Events popsánou v kapitole č. 3.5 je tohle řešení jednodušší i méně náročné na výpočetní zdroje, protože zde nedochází k neustálému ukládání událostí do souboru, jeho následnému čtení a analýze velkého množství SQL příkazů. Avšak nevýhodou je větší časový úsek, který je nutný k tomu, aby tato metoda zachytila stejné množství různých SQL příkazů jako metoda založená na Extended Events popsaná v kapitole č. 3.5. Z tohoto důvodu bylo toto řešení později rozšířeno o metodu založenou na SQL Server Plan Cache. Podle nastavení při instalaci DMV se určí, zda se nainstaluje metoda založená na aktuálně zpracovávaných SQL příkazech nebo metoda založená na SQL Server Plan Cache. Jak vypadá zpracování SQL příkazů pomocí metod DMV lze vidět na obr. č. 10.



Obrázek 10: Průběh zpracování pomocí metod DMV

3.6.1 Uložená procedura „PG_RunningQueriesJob“

Je to uložená procedura, která podle nastavení při instalaci (příznak v proměnné „@FromCache“) bude mít určeno, jaká z metod DMV se použije pro získání SQL příkazů. Tato uložená procedura se spouští v pravidelných intervalech pomocí jobu. Uložená procedura „PG_RunningQueriesJob“ má pouze jeden parametr, a tím je příznak, jestli se při spuštění mají vypisovat ladicí informace. Při spuštění pomocí jobu se spouští bez jakýchkoliv parametrů. Využití této uložené procedury je zobrazeno na obr. č. 10. Detailnější pohled této uložené procedury je na obr. č. 11.



Obrázek 11: Detail vykonání uložené procedury „PG_RunningQueriesJob“

3.6.1.1 Metoda založená na aktuálně zpracovávaných SQL příkazech

Pokud je v proměnné „@FromCache“ hodnota 0, po spuštění uložené procedury „PG_RunningQueriesJob“ se provede SQL dotaz pro získání aktuálně vykonávaných SQL příkazů nad SQL databází. V SQL dotazu je opět nutné vyfiltrovat nežádoucí SQL příkazy, které nechceme získávat. Získané aktuálně běžící SQL příkazy se ukládají do pomocné tabulky „tmpStatementTab“ vytvořené v rámci uložené procedury, jak lze vidět na obr. č. 11. Z toho důvodu, že se do pomocné tabulky uloží pouze aktuálně zpracovávané SQL příkazy, je velmi malá šance, že by jedno zavolání SQL dotazu pro získání všech aktuláně zpracovávaných SQL příkazů získalo všechny SQL příkazy reprezentující typické vytížení. Z tohoto důvodu se provádí získávání aktuálně zpracovávaných SQL příkazů vícekrát opakovaně za sebou. Tímto se docílí větší úspěšnosti a rychlejšího získání SQL příkazů, které reprezentují typické vytížení SQL databáze. Počet opakování je nastaven na 30. Je to z toho důvodu, aby se zvýšila šance na zachycení co největšího množství aktuálně zpracovávaných SQL příkazů. Pokud by úspěšnost byla stále nízká, je zde možnost, aby si uživatel sám změnil počet opakování v rámci jednoho běhu této procedury.

Když už je zachytávání aktuálně zpracovávaných SQL příkazů do pomocné tabulky „@tmpStatementTab“ u konce, provede se kontrola, jestli není tabulka prázdná. Pokud je prázdná, vykonávaní uložené procedury „PG_RunningQueriesJob“ se ukončí. Jestliže tabulka není prázdná, provede se průchod touto tabulkou pomocí kurzoru a pro každý SQL příkaz uložený v této tabulce se zavolá CLR uložená procedura „PG_WorkLoad“, která provede zpracování SQL příkazu. Po ukončení průchodu tabulky pomocí kurzoru je zpracování u konce a uložená procedura „PG_RunningQueriesJob“ ukončí své vykonávání.

3.6.1.2 Metoda založená na SQL Server Plan Cache

Pokud je v proměnné „@FromCache“ hodnota 1, po spuštění uložené procedury „PG_RunningQueriesJob“ se pomocí SQL dotazu získají SQL příkazy z SQL Server Plan Cache, jak je možné vidět na obr. č. 11. Opět se provede vyfiltrování nežádoucích SQL příkazů, které nechceme získávat. Výsledek tohoto dotazu se použije jako zdroj dat pro kurzor, který projde všechny SQL příkazy získané pomocí tohoto SQL dotazu, a pro každý z nich zavolá CLR uloženou proceduru „PG_WorkLoad“, která provede zpracování SQL příkazu. Po ukončení kurzoru je zpracování u konce a uložená procedura „PG_RunningQueriesJob“ ukončí své vykonávání. Ve výpisu kódu č. 10 je zobrazen tento SQL dotaz pro získání SQL příkazů z SQL Server Plan Cache. Z důvodu úspory místa nejsou zde vypsány všechny podmínky filtrace. Jak lze vidět ve výpisu kódu. č. 10, tento SQL příkaz pro získání SQL příkazů z SQL Server Plan Cache je odlišný od SQL příkazu, který lze vidět ve výpisu kódu č. 8. Je to z toho důvodu, že zde je potřeba z SQL Server Plan Cache získat texty SQL příkazů, narozdíl od získání hash hodnot plánů SQL příkazů, které je znázorněno ve výpisu kódu č. 8.

```
SELECT LTrim(RTrim(n.value('(@StatementText)[1]', 'VARCHAR(4000)'))) AS
    sql_text, n.value('(@StatementType)[1]', 'VARCHAR(4000)') AS sql_type
FROM ( SELECT plan_handle, query_plan
   FROM ( SELECT DISTINCT plan_handle
      FROM sys.dm_exec_cached_plans WITH(NOLOCK)
           ) AS qs
   OUTER APPLY sys.dm_exec_query_plan(qs.plan_handle) tp
       INNER JOIN sys.databases d ON tp.dbid = d.database_id
   where d.name='##DATABASE_NAME##') as tab (plan_handle, query_plan)
CROSS APPLY query_plan.nodes('/ShowPlanXML/BatchSequence/Batch/Statements/*')
    AS q(n)
where n.value('(@StatementType)[1]', 'VARCHAR(4000)') in ('INSERT','UPDATE','
    DELETE','SELECT')
```

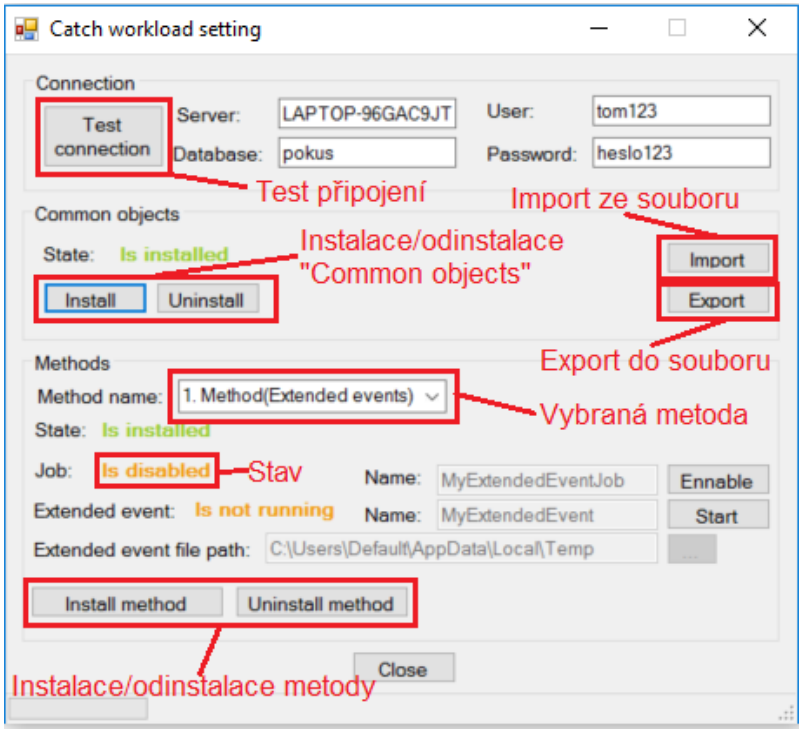
Výpis 10: Ukázka SQL dotazu pro získání SQL příkazů z SQL Server Plan Cache

3.6.2 Automatické spouštění procedur s pomocí Job

V tomto řešení bylo potřeba spouštět uloženou proceduru „PG_RunningQueriesJob“ popsanou v kapitole č 3.6.1 v pravidelných intervalech, a ideální k tomuto je využití SQL Agent a jobu. Co je to job, z čeho se skládá a jak funguje, je již popsáno v kapitolách 2.1.5 a 3.5.3. Zde bylo potřeba vytvořit obdobný job jako pro metodu založenou na Extended Events, ale s tím rozdílem, že tento job bude mít jiný název a také jiný plán spuštění. Pokud bylo při instalaci nastaveno, že se nainstaluje metoda založená na SQL Server Plan Cache, vytvoří se plán spuštění s intervalem co 5 minut. Jinak se vytvoří plán spuštění s intervalem co 10 sekund,protože jak již bylo zmíněno výše, různé plány spuštění jsou z toho důvodu, že čas potřebný k zachycení stejného množství SQL příkazů jako pomocí metody založené na Extended Events je větší, takže je nutné zvýšit počet spuštění jobu. Využití tohoto jobu je zobrazeno na obr. č. 10.

3.7 WorkLoad Add-In DLL knihovna

Je to knihovna psaná C#. Tato knihovna obsahuje třídu pro připojení a komunikaci s prostředím SQL Server Management Studio a také WinForm C# aplikaci, která se spouští v rámci SQL Server Management Studio jako Add-In. Tento Add-In je možné najít v záložce „Nástroje” pod názvem „WorkLoad Add-In settings”. Výsledný vzhled okna WorkLoad Add-In pro nastavení zachytávání typického vytížení nad SQL Server je na obr. č. 12.



Obrázek 12: Uživatelské rozhraní WorkLoad Add-In

3.7.1 Funkce implementované ve WorkLoad Add-In

Zde jsou vypsány funkce, které je možné využívat v rámci WorkLoad Add-In. Funkce jsou zde pouze uvedeny, jejich detailnější popis je uveden ve vlastní kapitole každé z funkcí.

- Instalace/odinstalace „Common objects" a metod DMV, Extended Events a nastavení těchto metod. Kapitola č. 3.7.2 a č. 3.7.3,
- Test připojení vůči SQL databázi (kapitola č. 3.7.4),
- Kontrola funkčnosti dřívějších instalací (kapitola č. 3.7.5),
- Export typického vytížení SQL Server do souboru (kapitola č. 3.7.6),
- Import typického vytížení SQL Server ze souboru (kapitola č. 3.7.7).

3.7.2 Instalace

Ve WinForm aplikaci WorkLoad Add-In je možné provést tři různé instalace. Instalace „Common objects" se provádí pomocí tlačítka „Install" (viz obr. č. 12), které je umístěno v části nazvané „Common objects". Tato instalace provádí spuštění skriptu „CreatePermissionAndAssembliesScript" pro povolení práv, aby bylo možné pracovat s CLR uloženými procedurami, a vytvoření assemblies pro práci s CLR uloženými procedurami. Dále provádí spuštění skriptu „CreateTableScript" pro vytvoření všech tabulek a jejich vazeb, které jsou popsány v kapitole č. 3. Nakonec se spouští skript „CreateCLRObjectScript" pro vytvoření CLR uložených procedur a funkcí. Volání všech skriptů se provádí v rámci jedné transakce. Pokud během transakce dojde k chybě, provede se zrušení všech změn pro celou transakci pomocí rollback a uživateli se vypíše chyba, která nastala. Po nainstalování je možné provádět export/import (viz kapitoly č. 3.7.6 a 3.7.7). Zda je instalace hotová, lze poznat podle stavu zobrazeného na obr. č. 12.

Druhá instalace se provádí pomocí tlačítka „Install method" (viz obr. č. 12), které je umístěno v části nazvané „Methods" a závisí ještě na hodnotě pole „Method name". Pokud je vybrána metoda „1. Method (Extended events)", provede se instalace metody založené na Extended Events. Během této instalace se spouští skripty „CreatePG_ExtendedEventsJob", „CreateExtendedEventsScript" a „CreateExtendedEventMethodJobScript". Spuštění skriptů je opět v rámci jedné transakce. Pokud dojde k chybě, provede se rollback transakce a uživateli se vypíše chyba. Před instalací je možné vybrat cestu k souboru, do kterého bude Extended Events ukládat. K výběru souboru slouží průzkumník. Dále je možné pojmenovat job a také samotnou Extended Events.

Po nainstalování již není možné měnit cestu ani názvy. Jakmile je instalace hotová, je možné pomocí tlačítek spouštět job a také zachytávat SQL příkazy pomocí Extended Events.

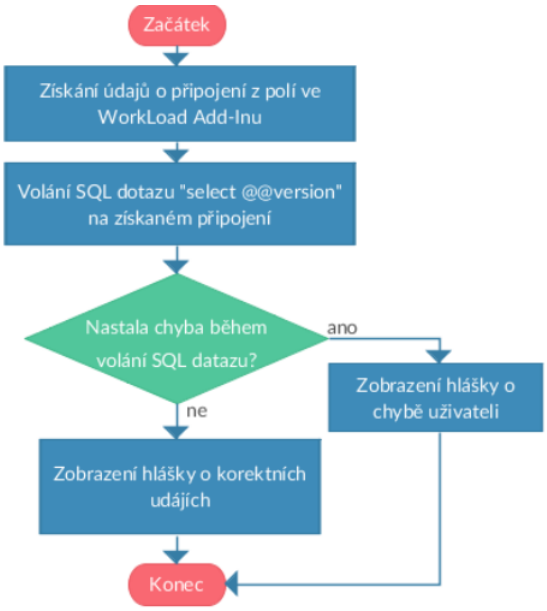
Třetí instalace se provádí stejně jako druhá instalace. Rozdíl je pouze v metodě, která se instaluje. U této instalace se instaluje metoda „2. Method (DVM)“. Před instalací je možné vybrat název jobu, a dále, zda se nainstaluje metoda založené na aktuálně zpracovávaných SQL příkazech nebo metoda založená na SQL Server Plan Cache. Během instalace této metody se spouštějí skripty „CreatePG_RunningQueriesJob“ a „CreateRunningQueriesMethodJobScript“. Po instalaci již není možné nic měnit, pouze lze spouštět job. Kdyby bylo potřeba cokoli měnit, musí se provést odinstalace a pak lze již měnit potřebné informace.

3.7.3 Odinstalace

Odinstalace probíhá obdobně jak bylo popsáno v kapitole č. 3.7.2, Jediná změna je v názvech skriptů, které se volají. Během instalace se volaly skripty s předponou „Create“, zatímco během odinstalace se volají skripty s předponou „Delete”. Tlačítka pro odinstalace jsou zobrazená na obr. č. 12.

3.7.4 Test připojení vůči SQL databázi

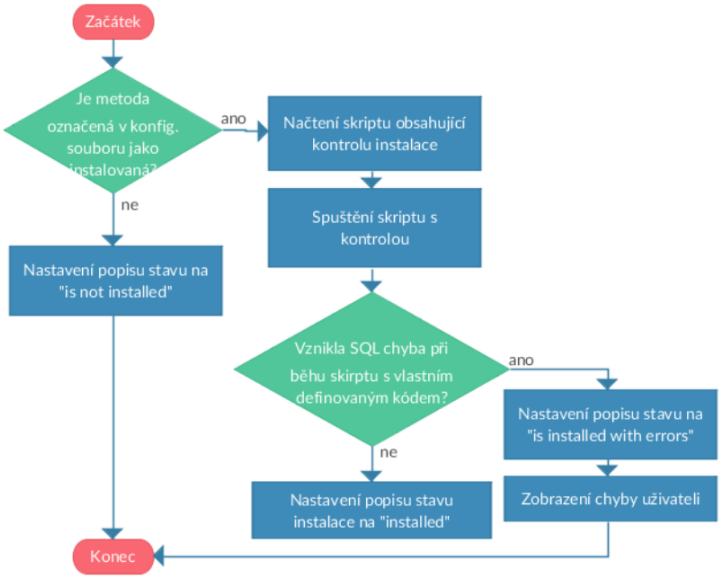
Pomocí testu lze zjistit, zda přihlašovací údaje zadané ve WorkLoad Add-In pro danou SQL databázi jsou správné a korektní. Tlačítko pro test funkčnosti připojení je zobrazeno na obr. č. 12. Na obr. č. 13 je zobrazen postup při testování funkčnosti připojení na SQL databázi.



Obrázek 13: Vývojový diagram zobrazující postup při testování připojení na SQL databázi

3.7.5 Kontrola funkčnosti dřívějších instalací

Nastavení WorkLoad Add-In se během práce s ním ukládá do jeho konfiguračního souboru. Díky tomu je po opětovném spuštění vše nastaveno tak, jak to bylo před ukončením práce s WorkLoad Add-In. Po jeho spuštění je tedy možné zjistit, které instalace jsou aktuálně nainstalovány z konfiguračního souboru. Podle toho je možné zobrazit stav instalace, jak lze vidět na obr. č. 12. Může se však stát, že někdo provede změnu v SQL databázi ručně, tímpádem se to WorkLoad Add-In nedozví. Poté by po spuštění zobrazoval např. to, že je metoda založená na Extended Events nainstalována, i když to ve skutečnosti nemusí být pravda. Každá z instalací popsaných v kapitole č. 3.7.2 má svůj vlastní skript uložený v datech WorkLoad Add-In. V tomto skriptu jsou SQL příkazy pro kontrolu, zda je metoda instalována. Pokud nějaká kontrola neprojde, pak vznikne chyba s daným kódem. Tento kód společně s hláškou je poté zobrazen uživateli. Kontrola se provádí po spuštění WorkLoad Add-In pro všechny instalace. Průběh kontroly lze vidět obr. č. 14.

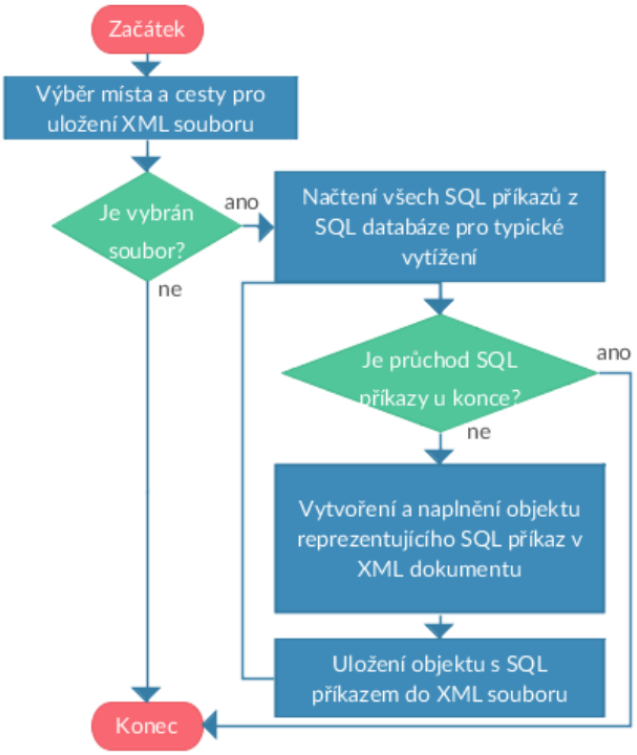


Obrázek 14: Vývojový diagram zobrazující obecný postup při kontrole, zda je metoda instalována

3.7.6 Export typického vytížení SQL Serveru do XML souboru

Pomocí exportu je možné vyexportovat typické vytížení SQL Server z SQL databáze do XML souboru. Tlačítko pro export je zobrazeno na obr. č. 12.

Export funguje na principu přečtení všech příkazů uložených v SQL databázi s typickým vytížením a jeho následným uložením do XML souboru. XML soubor musí mít specifickou strukturu, aby bylo možné do něj vhodně uložit typické vytížení SQL Server. Jak vypadá struktura XML souboru je možné vidět ve výpisu č. 15. Aby bylo možné provádět export do XML souboru, je nutné mít nainstalovány „Common Objects“, které jsou popsány v kapitole č. 3.7.2. Průběh exportu SQL příkazů do XML souboru je zobrazen na obr. č. 15.

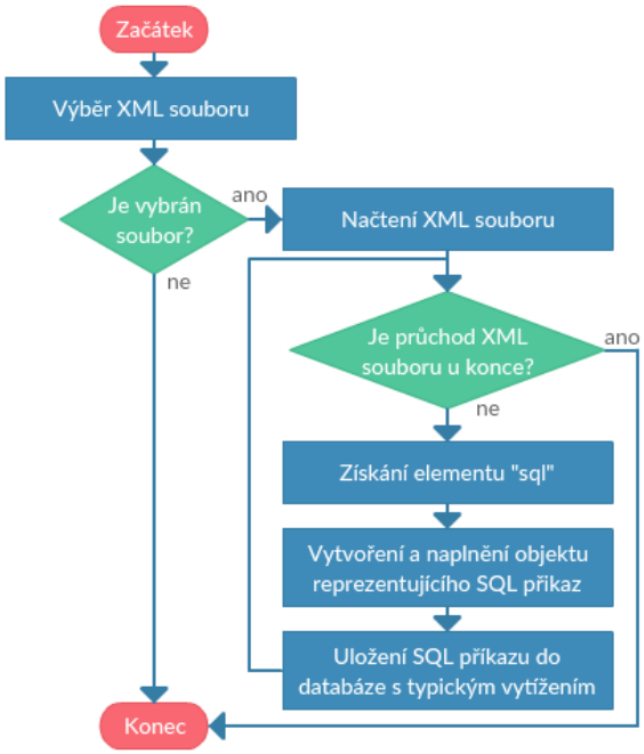


Obrázek 15: Vývojový diagram pro export typického vytížení SQL Server do XML souboru

3.7.7 Import typického vytížení SQL Serveru z XML souboru

Pomocí importu je možné naimportovat typické vytížení SQL Server z XML souboru do SQL databáze s typickým vytížením. Import funguje na principu přečtení XML souboru s typickým vytížením SQL Server a jeho uložením do SQL databáze. Tlačítko pro import je zobrazeno na obr. č. 12. Před samotným uložením SQL příkazu se provádí kontrola, zda SQL příkaz již v SQL

databázi neexistuje. Pokud již SQL příkaz existuje, přidají se pouze parametry tohoto příkazu. Jestliže ale SQL příkaz neexistuje, vloží se nový SQL příkaz i s jeho parametry. Struktura XML dokumentu musí být stejná jako na obr. č. 15. Pokud nebude struktura dodržená, dojde k chybě a uživateli bude vypsána chybová hláška. Detail, jak probíhá import, lze vidět na obr. č. 16.



Obrázek 16: Vývojový diagram pro import typického vytížení SQL Server z XML souboru

3.8 WorkLoad instalátor

Z důvodu zjednodušení instalace WorkLoad Add-In a jeho součástí na hostitelský počítač byla vytvořená aplikace pro instalaci a odinstalaci WorkLoad Add-In a jeho potřebných konfiguračních souborů a souborů, ve kterých jsou skripty pro vytváření i mazání tabulek, uložených procedur, Extended Events i připojení assemblies k instanci SQL Server. Popis práce s instalátorem je popsán v uživatelské příručce v elektronické příloze na CD/DVD písm. A.

4 Optimalizace

V této kapitole jsou popsány optimalizace, které jsou implementovány, aby co nejvíce minimalizovaly zatížení systému v důsledku neefektivního zachytávání typického vytížení. Budou zde popsány optimalizace, které snižují vytížení SQL Server, ale také optimalizace, které pomáhají k rychlejšímu získání typického vytížení SQL databáze.

4.1 Opakované čtení z tabulky aktuálně zpracovávaných SQL příkazů

Tato optimalizace je v uložené proceduře „PG_RunningQueriesJob“ vytvořená z toho důvodu, že během jednoho přečtení aktuálně zpracovávaných SQL příkazů nebude v daný okamžik zachyceno mnoho SQL příkazů. Z tohoto důvodu se toto čtení provede vícekrát za sebou. Tímto se docílí většího množství zachycených SQL příkazů během jednoho spuštění. Opakované čtení je implementováno pomocí cyklu WHILE. V tomto cyklu se opakovaně čte z tabulky aktuálně zpracovávaných SQL příkazů a tyto příkazy se vkládají do pomocné tabulky. Jakmile skončí čtení pomocí cyklu, projde se tato pomocná tabulka pomocí kurzoru a SQL příkazy v ní uložené se zpracují.

4.2 Čtení z Extended Events souboru

Tato optimalizace je v uložené proceduře „PG_ExtendedEventsJob“ vytvořená z toho důvodu, že původní řešení bylo náročnější na vykonání. Původní načítaní ze souboru je uvedeno ve výpisu kódu č. 11. Zde je problém s konverzí datového typu atributu „event_data“ na XML. Tento převod je časově náročný vzhledem k tomu, že se tato konverze datového typu musí provést pro každý SQL příkaz uložený v souboru. Další nevýhodou je následné vyhledávání pomocí XPath v atributu „event_data“.

```
select event_data.value('(event/data[@name="statement"])[1]', 'nvarchar(max)')
    AS sql_text
from(
   select cast(event_data as xml) event_data,file_offset
   FROM sys.fn_xe_file_target_read_file(@filePath, NULL, NULL, NULL)
)t
```

Výpis 11: Ukázka čtení z Extended Events souboru před optimalizaci

Nové řešení funguje na principu získávání SQL příkazu přímo z atributu „event_data” bez konverze datového typu. Jak to tedy funguje? Je to jednoduché. Nejprve byla provedená analýza, jak vypadá obsah atributu „event_data”. V tomto atributu je text ve formátu XML. V něm se najde

pomocí funkce „CHARINDEX" index začátku SQL příkazu, který je v atributu „StrStart", a index konce SQL příkazu. Nakonec stačí v atributu „event_data_str" pomocí funkce „SUBSTRING" najít text, který je mezi těmito indexy. Tímto se získá text SQL příkazu bez nutnosti konverze na datový typ XML. Pro pochopení popisu optimalizace jsem zde přidal výpis kódu č. 12, kde jde vidět popisovaná optimalizace.

```
SELECT distinct SUBSTRING(event_data_str, StrStart, StrStop - StrStart)
FROM
( SELECT [object_name], event_data AS event_data_str
   , CHARINDEX('<data name="statement"><value><![CDATA[', event_data) + LEN('<
      data name="statement"><value><![CDATA[') AS StrStart
   , CHARINDEX(']]></value>', event_data, CHARINDEX('<data name="statement"><
      value><![CDATA[', event_data) + LEN('<data name="statement"><value><![
      CDATA[') + 1) AS StrStop
  FROM sys.fn_xe_file_target_read_file(@filePath, NULL, NULL, NULL
)
```

Výpis 12: Ukázka čtení z Extended Events souboru po optimalizaci

4.3 Zrušení pomocné tabulky

Tato optimalizace je v „PG_ExtendedEventsJob". Jak již samotný název napovídá, zde bylo výhodnější zrušit ukládání SQL příkazů Extended Events souboru do pomocné tabulky, jak bylo vytvořeno v původním návrhu. Díky tomu, že se nemusí ukládát SQL příkazy ze souboru do pomocné tabulky, která se stejně jenom prochází pomocí kurzoru, je zpracování rychlejší. Je to z důvodu režie ukládání a následného čtení z pomocné tabulky. Po zrušení ukládání do pomocné tabulky se SQL dotaz pro získání SQL příkazů ze souboru rovnou použije pro průchod pomocí kurzoru. Implementace lze vidět na obr. č. 16.

4.4 Optimalizace kurzoru

Když se vytváří kurzor, je možné u něj definovat argumenty, které ovlivňují jeho chování během zpracování. Jako optimalizaci je zde použity argument „FAST_FORWARD". Tento argument způsobí stejné chování kurzoru jako kdyby se použily argumenty „READ_ONLY" a „FORWARD_ONLY". Argument „READ_ONLY" zabraňuje aktualizacím v kurzoru. Argument „FORWARD_ONLY" říká, že kurzor může procházet výsledky SQL dotazu pouze dopředu. Díky těmto argumentům má kurzor povolené výkonnostní optimalizace. Jak vypadá kurzor po optimalizaci lze vidět ve výpisu kódu č. 16.

4.5 Čtení z SQL Server Plan Cache před zpracováním Extended Events souboru

Tato optimalizace je v uložené proceduře „PG_ExtendedEventsJob" popsané v kapitole č. 3.5.2. Před tím, než se provedla tato optimalizace, fungovala uložená procedura „PG_ExtendedEventsJob" následovně. Během každého spuštění se provedlo čtení z Extended Events souboru. Přečtené SQL příkazy se pokaždé všechny procházely a každý SQL příkaz byl analyzován pomocí CLR uložené procedury „PG_WorkLoad" popsané v kapitole č. 3.3.1. Zde však může nastat situace, kdy v Extended Events souboru nejsou žádné nové SQL příkazy, které by již nebyly zachyceny v tabulkách typického vytížení. I když nastane taková situace, Extended Events soubor se přesto zpracovává celý a tím zbytečně zatěžuje SQL Server. Na obr. č. 9 je zobrazeno využití této optimalizace.

Optimalizace spočívá v tom, že před samotným čtením Extended Events souboru a jeho zpracováním se provede nahlédnutí do SQL Server Plan Cache. Od každého z těchto plánů se vezme jejich hash hodnota, pro kterou se provede kontrola, zda již existuje v tabulce „TB_SqlCacheHash". Pokud taková hash hodnota ještě v tabulce není, vloží se do této tabulky. Po čtení z SQL Server Plan Cache se kontroluje, zda v tabulce „TB_SqlCacheHash" přibyly nějaké nové hash hodnoty. Pokud nepřibyly žádné nové hash hodnoty, není nutné načítat SQL příkazy z Extended Events tabulky. Z počátku se tato optimalizace může jevit jako zhoršení, protože bude nutné přidávat nové hash hodnoty do tabulky „TB_SqlCacheHash" a zároveň číst SQL příkazy z Extended Events souboru. Ale po několika spuštěních se začne výskyt nových hash hodnot, které ještě nejsou v tabulce „Tb_SqlCacheHash", snižovat a tímpádem se začne snižovat počet čtení z Extended Events souboru. Jak takové čtení z SQL Server Plan Cache vypadá je uvedeno na výpisu kódu č. 10.

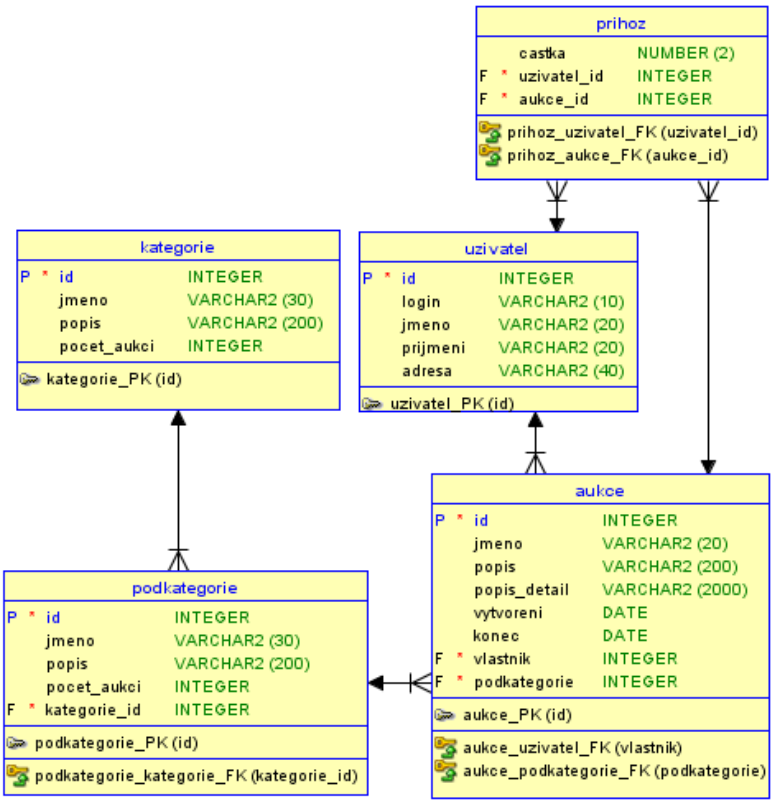
5 Testování

V této kapitole je popsáno testováním vytvořeného nástroje pro zachycení typického vytížení SQL Server, který je popsán v kapitole č. 3. Testováním je myšleno monitorování chodu nástroje a zjištění, jak moc je tento nástroj náročný na zdroje, neboli jak moc vytěžuje systém z hlediska vytížení procesoru.

Testování i samotný vývoj celého řešení probíhalo na osobním notebooku, který má procesor Intel Core i3 o taktu 2.0 GHz s integrovanou grafickou kartou, velikost RAM paměti 8 GB a velikost HDD 1 000 GB. Na notebooku je nainstalován operační systém Microsoft Windows 10. Jako databázové prostředí byl použit Microsoft SQL Server 2016 Developer Edition (verze 13.0.1601.5), který obsahuje SQL Agent. Nakonec pro přístup na databázi v SQL Server bylo využito nástroje Microsoft SQL Server Management Studio 2012 (verze 11.0.2100.60).

Pro monitorování využití zdrojů byl použit nástroj Sledování výkonu systému Windows (Performace monitor), který je součástí operačního systému Windows. Tento nástroj slouží k monitorování programů, které běží na počítači, a zobrazení grafu vytížení zdrojů počítače, a to v reálném čase pomocí sběru dat. Využívá k tomu výkonnostní počítadla, stopy událostí dat a konfigurační informace z registrů Windows. Pro zachycení vytížení procesoru byl využit čítač s názvem „% času procesoru“. Během měření i po jeho skončení je možné vidět dobu trvání měření, nejvyšší a nejnižší dosaženou hodnotu během měření a průměrnou hodnotu za celkovou dobu měření. V tomto případě byla zaznamenána pouze průměrná hodnota za celkovou dobu měření. Během měření byly sledovány dva procesy, a to „sqlservr.exe“, ve kterém je spuštěn SQL Server a „sqlagent.exe“, což je služba, která provádí spouštění naplánovaných jobů vytvořených v SQL Server.

Pro testování vytížení byla využitá konzolová aplikace, která se využívá pro vytěžování vzorové databáze v rámci předmětu Administrace databázových systémů. Tato konzolová aplikace posílá 11 různých SQL příkazů na SQL databázi a tím tuto SQL databázi vytěžuje. Tato konzolová aplikace mimo jiné obsahuje také skript pro vytvoření testovacích tabulek v SQL databázi a skript pro vytvoření počátečních testovacích dat. Konzolová aplikace provádí simulaci aukční síně. Jsou zde definovány tabulky pro uživatele, aukce, příhozy, kategorie a podkategorie. Struktura databáze je zobrazená na obr. č. 17.



Obrázek 17: Schéma tabulek pro simulaci aukční síně

V tabulce č. 1 je vypsáno množství záznamů v jednotlivých databázových tabulkách a také množství využitého místa, které tyto tabulky zabírají v SQL databázi. Tyto údaje byly zaznamenány před samotným spuštěním konzolové aplikace pro simulaci vytížení nad SQL databází.

Název tabulky	Počet záznamů	Velikost využitého místa [KB]	Velikost nevyužitého místa [KB]	Celková velikost [KB]
aukce	179154	37576	192	37768
kategorie	4	16	56	72
podkategorie	10	16	56	72
prihoz	100000	2096	152	2248
uzivatel	99001	6864	184	7048

Tabulka 1: Využití místa v SQL databázi pro tabulky

Aplikace pro simulaci vytížení byla spouštěná ve všech testovacích případech vždy se stejnými parametry, takže podmínky simulace byly pro všechny testy shodné. Nastavení parametrů spuš-

tění aplikace přes příkazový řádek je zobrazeno na výpisu č. 13. Popis parametrů konzolové aplikace je uveden v příloze písm. E.

```
java -jar Aukce_SQLServer_WaitWorkload.jar 0.01 1 700 LAPTOP-104LA4TB bau0014
    user password
```

Výpis 13: Ukázka nastavení parametrů při spouštění testovacího vytížení

Pro každý testovací případ se provádělo spuštění aplikace pro simulaci vytížení vždy 5krát a každé měření trvalo 10 minut. Z každého běhu se získaly průměrné hodnoty a pro tyto hodnoty se vypočítal celkový průměr za všechny spuštění, čímž se získá celkové vytížení procesoru pro jeden testovací případ. Počet spuštění aplikace pro simulaci vytížení byl pro každý testovací případ stejný.

5.1 Testování a výsledky pro metodu založenou na Extended Events

Zde se provádělo testování vytížení procesoru počítače pro metodu založenou na Extended Events popsanou v kapitole č. 3.5.

Testovací případ	Hodnota průměru pro „sqlservr.exe“	Hodnota průměru pro „sqlagent.exe“
Base line	21.199	0.003
Bez optimalizace a se zpracováním SQL příkazů	27.872	0.011
S optimalizací a se zpracováním SQL příkazů	24.529	0.015

Tabulka 2: Využití „% času procesoru“ pro testovací případy metody založené na Extended Events se zpracováním SQL příkazů

Base line (viz tab. č. 2) - u tohoto testovacího případu se provádí pouze vytížení pomocí konzolové aplikace popsané v kapitole č. 5. Tento případ je zde zahrnut, aby bylo možné si udělat představu, o kolik jednotlivé testovací případy vytěžují procesor více než samotné vytížení nad SQL databázi.

Bez optimalizace a se zpracováním SQL příkazů (viz tab. č. 2) - u tohoto testovacího případu se provádí testování se zpracováním SQL příkazů pomocí CLR uložené procedury „PG_WorkLoad“ popsané v kapitole č. 3.3.1, avšak bez využití optimalizace pomocí SQL Server Plan Cache popsané v kapitole č. 4.5. To znamená, že v tomto testovacím případu se tedy provádí ukládání do Extended Events souboru a následné čtení a zpracování SQL příkazů z něj. Jak lze vidět v tabulce č. 2, tento testovací případ dopadl nejhůře ze všech, protože zde dochází ke zpracování SQL příkazů a nebyla zde využitá optimalizace pomocí SQL Server Plan Cache.

S optimalizací a se zpracováním SQL příkazů (viz tab. č. 2) - u tohoto testovacího případu se provádí testování se zpracováním SQL příkazů pomocí CLR uložené procedury „PG_WorkLoad“ popsané v kapitole č. 3.3.1 a zároveň s optimalizací pomocí SQL Server Plan Cache popsané v kapitole č. 4.5. To znamená, že v tomto testovacím případu se tedy provádí ukládání do Extended Events souboru a následné čtení a zpracování SQL příkazů z něj. Jak lze vidět v tabulce č.2, při využití optimalizace je pro tento testovací případ průměrná hodnota pro proces „sqlservr.exe“ výrazně nižší než u varianty, která optimalizaci nepoužívá. U tohoto testovacího případu bylo provedeno ověření, kolikrát se musí zpracovat Extended Events soubor, než zafunguje optimalizace pomocí SQL Server Plan Cache. Po ověření bylo zjištěno, že už od třetího spuštění uložené procedury „PG_ExtendedEventsJob“ popsané v kapitole č. 3.5.2 zafunguje optimalizace a Extended Events soubor se nebude zpracovávat.

V tabulce č. 3 jsou uvedeny dva testovací případy, které slouží pouze pro znázornění, jak moc se liší vytížení procesoru bez zpracování SQL příkazů a se zpracováním SQL příkazů (viz tabulka č. 2).

Testovací případ	Hodnota průměru pro „sqlservr.exe“	Hodnota průměru pro „sqlagent.exe“
Bez optimalizace a zpracování SQL příkazů	24.226	0.023
S optimalizací a bez zpracování SQL příkazů	22.356	0.011

Tabulka 3: Využití „% času procesoru“ pro testovací případy metody založené na Extended Events bez zpracování SQL příkazů

Bez optimalizace a zpracování SQL příkazů (viz tab. č. 3) - u tohoto testovacího případu se provádí testování bez volání CLR uložené procedury „PG_WorkLoad“ popsané v kapitole č. 3.3.1 a bez optimalizace pomocí SQL Server Plan Cache popsané v kapitole č. 4.5. To znamená, že v tomto testovacím případu se tedy pouze provádí ukládání do Extended Events souboru a následné čtení SQL příkazů z něj, přičemž se s SQL příkazy nic nedělá (neprovádí se jejich zpracování). Je zde tedy zachyceno pouze vytížení samotné procedury „PG_ExtendedEventsJob“ popsané v kapitole č. 3.5.2 bez výše zmíněné optimalizace.

S optimalizací a bez zpracování SQL příkazů (viz tab. č. 3) - u tohoto testovacího případu se provádí testování opět bez volání CLR uložené procedury „PG_WorkLoad“ popsané v kapitole č. 3.3.1 ale s optimalizací pomocí SQL Server Plan Cache popsané v kapitole č. 4.5. To znamená, že v tomto testovacím případu se tedy pouze provádí ukládání do Extended Events souboru a následné čtení SQL příkazů z něj, přičemž se s SQL příkazy nic nedělá (neprovádí se jejich zpracování). Je zde tedy zachyceno pouze vytížení samotné procedury „PG_ExtendedEventsJob“ popsané v kapitole č. 3.5.2 s výše zmíněnou optimalizací.

Vzhledem k tomu, že tato metoda provádí zachycení všech SQL příkazů volaných v SQL databázi, je úspěšnost zachycení všech typů SQL příkazů velmi vysoká. Již při prvním zpracování Extended Events souboru bylo zachyceno všech 11-ti typů SQL příkazů, které posílá na SQL databázi konzolová aplikace pro simulaci vytížení.

5.2 Testování a výsledky pro metody DMV

Zde se provádělo testování vytížení procesoru pro metody DMV popsané v kapitole č. 3.6.

Testovací případ	Hodnota průměru pro „sqlservr.exe“	Hodnota průměru pro „sqlagent.exe“
Metoda založená na aktuálně zpracovávaných SQL příkazech	22.150	0.022
Metoda založená na SQL Server Plan Cache	23.050	0.021

Tabulka 4: Tabulka s využitím „% času procesoru“ pro testovací případy metod DMV

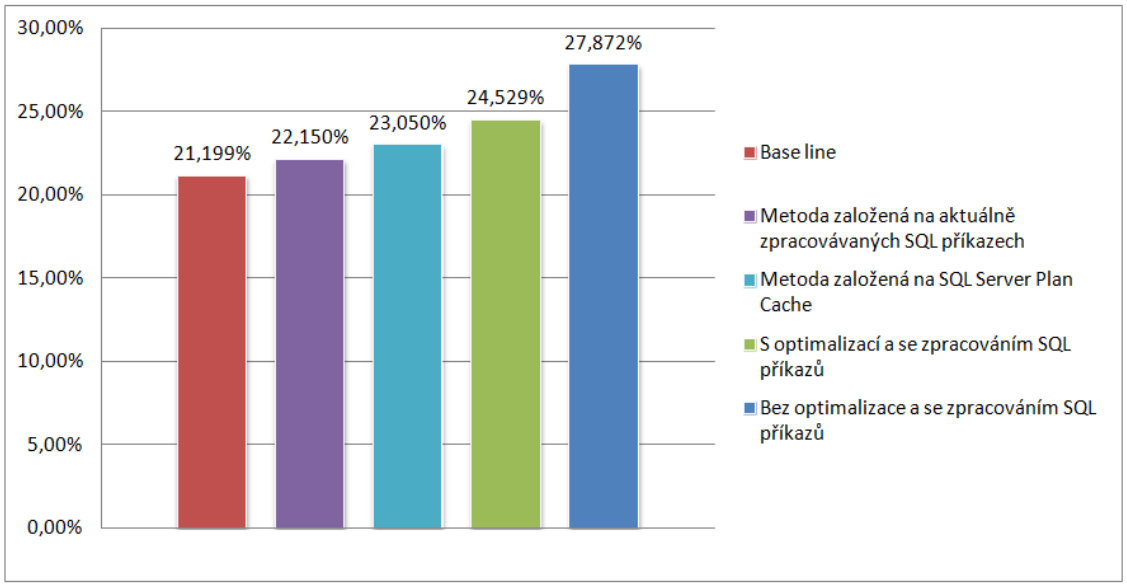
Metoda založená na aktuálně zpracovávaných SQL příkazech (viz tab. č. 4) - u tohoto testovacího případu se provádí testování metody založené na aktuálně zpracovávaných SQL příkazech popsané v kapitole č. 3.6.1.1. Jak je možné vidět v tabulce č. 4, tento testovací případ má průměr vytížení procesem „sqlservr.exe“ o trochu nižší než testovací případ metody založené na SQL Server Plan Cache, a to z důvodu popsaného v kapitole č. 3.6.1.1. Výkonnostně je to výhodnější, ale aby bylo možné zachytit stejné množství jako pomocí testovacího případu metody založené na SQL Server Plan Cache, je nutné vykonat více běhů.

Metoda založená na SQL Server Plan Cache (viz tab. č. 4) - u tohoto testovacího případu se provádí testování metody založené na SQL Server Plan Cache popsané v kapitole č. 3.6.1.2. Jak je možné vidět v tabulce č. 4, tento testovací případ má průměr vytížení procesem „sqlservr.exe“ o trochu vyšší, ale na druhou stranu zachycení typického vytížení je rychlejší než v testovacím případě metody založené na aktuálně zpracovávaných SQL příkazech.

Úspěšnost zachycení všech typů SQL příkazů je u metody založené na SQL Server Plan Cache velmi vysoká. Již při prvním čtení z SQL Server Plan Cache došlo k získání všech 11-ti typů SQL příkazů, které posílá na SQL databázi konzolová aplikace pro simulaci vytížení. Avšak metoda založená na aktuálně zpracovávaných SQL příkazech již takovou úspěšnost neměla. Během 10-ti minut testování tato metoda dokázala získat pouze 9 z 11-ti SQL příkazů, které posílá na SQL databázi konzolová aplikace pro simulaci vytížení.

5.3 Shrnutí

Pokud porovnáme výsledky metod Extended Events a DMV (viz tab. č. 2 a 4), rozdíly ve vytížení procesoru mezi těmito metodami nejsou nijak výrazné, jak je možné vidět v grafu na obr. č. 18. Každá z těchto metod má své výhody i nevýhody, které byly popsány v kapitole č. 2.2, přičemž podle těchto výhod a nevýhod je zřejmé, že každá metoda se hodí pro jinou situaci. Výběr z těchto metod závisí na konkrétním případu použití.



Obrázek 18: Graf zobrazující využití „% času procesoru“ procesu „sqlservr.exe“ pro testovací případy (viz tab. č. 2 a 4)

6 Závěr

Cílem této diplomové práce bylo navrhnout a implementovat nástroj, který umožní zachycení typického vytížení nad SQL Server. Vytvořený nástroj umožní dle nastavení automaticky sbírat informace o provedených SQL příkazech a za běhu provádět jejich analýzu a uložení do SQL databáze zachycující typické vytížení.

Teoretická část této diplomové práce byla věnována popisu a analýze nástrojů a možných řešení zachycení typického vytížení nad SQL Server. U každého přípustného řešení byly popsány obecné kroky vedoucí k zachycení a uložení typického vytížení do SQL databáze a bylo provedeno zhodnocení výhod a nevýhod jednotlivých metod.

Praktická část diplomové práce byla věnována návrhu a implementaci tří metod pro zachycení, zpracování a uložení typického vytížení do SQL databáze. Také je zde popsána integrace implementovaných metod do SQL Server Management Studio pomocí Add-In rozšíření. V praktické části jsou také popsány optimalizace, které pomohly snížit využití zdrojů počítače při zachytávání typického vytížení, a to vše bylo řádně otestováno a výsledky byly porovnány a zhodnoceny.

Hlavním přínosem diplomové práce je pro mě osvojení a zdokonalení se ve využití nástrojů a technologií, s kterými jsem se doposud nesetkal. Získané znalosti během tvorby této diplomové práce a také širší přehled považuji za velký přínos i do budoucího zaměstnání.

Literatura

[1] Josef Finsel, *How Can SQL Server Agent Make Life Easier?*, The Handbook for Reluctant Database Administrators, [cit. 2017-02-18], pages 331-256, ISBN 978-1-4302-1146-4, vyd. Apress, 2001

[2] Janis Griffin, *Improve SQL Server Performance Management with Extended Events - A guide to getting started* [online], [cit. 2017-02-18]. Dostupné z: (http://cdn.swcdn.net/creative/pdf/Whitepapers/SQLServer_ExtendedEvents_WP_Confio_May2014.pdf)

[3] Microsoft®, *SQL Trace* [online], [cit. 2017-02-18]. Dostupné z: (https://docs.microsoft.com/en-us/sql/relational-databases/sql-trace/sql-trace)

[4] Microsoft®, *Events notification* [online], [cit. 2017-02-18]. Dostupné z: (https://docs.microsoft.com/en-us/sql/relational-databases/service-broker/event-notifications)

[5] Mike Wachal, *Extended events reader* [online], [cit. 2017-02-18]. Dostupné z: (https://blogs.msdn.microsoft.com/extended_events/2011/07/20/introducing-the-extended-events-reader/)

[6] Steve Stein, Jim Vance, Craig Guyer, *SQL Server Agent* [online], [cit. 2017-02-18]. Dostupné z: (https://docs.microsoft.com/en-us/sql/ssms/agent/sql-server-agent)

[7] Microsoft®, *Trigger* [online], [cit. 2017-02-18]. Dostupné z: (https://msdn.microsoft.com/en-us/library/sdk3bcyw.aspx)

[8] Microsoft®, *CLR Stored Procedures* [online], [cit. 2017-02-18]. Dostupné z: (https://msdn.microsoft.com/en-us/library/ms131094.aspx)

[9] Rick Byham, Craig Guyer, *Dynamic management views* [online], [cit. 2017-02-18]. Dostupné z: (https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/system-dynamic-management-views)

[10] Thomas Russ, *Add-In* [online], [cit. 2017-02-18]. Dostupné z: (https://sqljudo.wordpress.com/31-days-of-sql-server-management-studio/ssms-day-27-building-a-custom-add-in-extension/)

[11] Jonathan Kehayias, *Measuring „Observer Overhead" of SQL Trace vs. Extended Events* [online], [cit. 2017-02-18]. Dostupné z: (https://sqlperformance.com/2012/10/sql-trace/observer-overhead-trace-extended-events)

A CD/DVD

- Uživatelská příručka pro instalaci WorkLoad Add-In
- Instalátor WorkLoad Add-In a k němu potřebné soubory
- Zdrojové soubory WorkLoad Add-In a instalátoru

B Skript pro vytvoření Extended Events

```
CREATE EVENT SESSION ##EXTENDED_EVENT_NAME## ON SERVER
ADD EVENT sqlserver.sp_statement_completed(SET collect_statement=(1)
  ACTION(sqlserver.is_system)
  WHERE ( [sqlserver].[is_system]=(0)
    AND (([statement] like 'INSERT%' OR [statement] like 'UPDATE%'
      OR [statement] like 'DELETE%' OR [statement] like 'SELECT%' ))
    AND [sqlserver].[database_name]=N'##DATABASE_NAME##'
    AND [statement] not like '%tb_query%'
    AND [statement] not like '%tb_param%'
    AND [statement] not like '%filePathTable%')),
ADD EVENT sqlserver.sql_statement_completed(SET collect_statement=(1)
    ACTION(sqlserver.is_system)
    WHERE ( [sqlserver].[is_system]=(0)
    AND (([statement] like 'INSERT%' OR [statement] like 'UPDATE%'
      OR [statement] like 'DELETE%' OR [statement] like 'SELECT%' ))
    AND [sqlserver].[database_name]=N'##DATABASE_NAME##'
    AND [statement] not like '%tb_query%'
    AND [statement] not like '%tb_param%'
    AND [statement] not like '%tmpStatementTab%'))
ADD TARGET package0.event_file(SET filename=N'##EXTENDED_EVENT_FOLDER_PATH##
    Query1.xel')
WITH (MAX_MEMORY=4096 KB,EVENT_RETENTION_MODE=NO_EVENT_LOSS,
    MAX_DISPATCH_LATENCY=1 SECONDS,MAX_EVENT_SIZE=0 KB,MEMORY_PARTITION_MODE=
    NONE,TRACK_CAUSALITY=OFF,STARTUP_STATE=OFF)
```

Výpis 14: Ukázka skriptu pro vytvoření Extended Events

C Struktura XML s typickým vytížením SQL Server

```
<?xml version="1.0" encoding="utf-8"?>
<workload>
  <sql id="1" paramcount="1" count="1" type="dql" text="SELECT * FROM AUKCE T0
      WHERE T0.ID = @VAR1">
    <values>
      <p>10</p>
    </values>
  </sql>
  <sql id="2" paramcount="0" count="1" type="dql" text="SELECT @@TRANCOUNT" />
  <sql id="3" paramcount="1" count="2" type="dql" text="SELECT T0.NAME FROM SYS
      .DATABASE_PRINCIPALS T0 WHERE T0.TYPE = @VAR1">
    <values>
      <p>'A'</p>
    </values>
    <values>
      <p>'R'</p>
    </values>
  </sql>
</workload>
```

Výpis 15: Ukázka struktury XML s typickým vytížením SQL Server

D Skript pro čtení a zpracování SQL příkazů z Extended Events souboru

```
DECLARE db_cursor CURSOR FAST_FORWARD FOR
SELECT distinct SUBSTRING(event_data_str, StrStart, StrStop - StrStart)
FROM    ( SELECT   [object_name],
     event_data AS event_data_str
   , CHARINDEX('<data name="statement"><value><![CDATA[', event_data)
         + LEN('<data name="statement"><value><![CDATA[') AS StrStart
   , CHARINDEX(']]></value>', event_data,
           CHARINDEX('<data name="statement"><value><![CDATA[', event_data)
           + LEN('<data name="statement"><value><![CDATA[') + 1) AS StrStop
     FROM     sys.fn_xe_file_target_read_file(@filePath, NULL, NULL, NULL)
   ) AS K
OPEN db_cursor
FETCH NEXT FROM db_cursor INTO @sqlText

WHILE @@FETCH_STATUS = 0
BEGIN
  EXECUTE dbo.PG_WorkLoad @sqlQuery=@sqlText ,@error=@error OUTPUT

  IF CHARINDEX('success',@error) AND @Debug=1 > 0
    SELECT @error
  SET @i=@i+1
  FETCH NEXT FROM db_cursor INTO @sqlText
END
CLOSE db_cursor
DEALLOCATE db_cursor
```

Výpis 16: Ukázka kurzoru pro čtení a zpracování SQL příkazů z Extended Events souboru

E Popis parametrů a spuštění konzolové aplikace pro simulaci vytížení

Konzolová aplikace se spouští z přikazového řádku s parametry, které ovlivňují běh. Těchto parametrů je hned několik, což umožňuje široké nastavení pro různé potřeby využití.

Parametry aplikace:

- cekani – přibližná doba čekání v sekundách mezi jednotlivými požadavky posílanými uživatelem,
- uzivatelu – počet paralelních vytvořených vláken, která budou dotazovat server,
- dobabehu – přibližný čas v sekundách, po který bude jedno vlákno posílat SQL příkazy,
- host – IP adresa nebo hostname serveru,
- db – jméno databáze, ke které se bude aplikace připojovat,
- uziv – jméno uživatele použitelného při přihlášení,
- pass – heslo tohoto uživatele.